1. AngularJS Product Catalog App

Q1: What are AngularJS filters?

Main Answer:

AngularJS filters are used to format data before displaying it in the view.

Detailed Explanation:

Filters allow you to transform the output in HTML, such as changing text to uppercase, sorting lists, formatting numbers, or converting currency values. They help improve readability without changing the original data in the model. Common filters include uppercase, lowercase, currency, number, orderBy, and filter. Filters are applied using the pipe (|) symbol.

How do filters affect data binding?

Filters only affect how the data looks in the view—they do not change the actual data in the model. This means your data stays clean and unmodified in the backend or controller, but it's presented in a user-friendly way in the frontend.

Can you create custom filters? How?

Yes, AngularJS allows custom filters. You can define them using app.filter() in your Angular module. For example, if you want to format a date or string in a special way that default filters don't provide, you can write your own logic and use it like a built-in filter.

What is the syntax to apply a filter in HTML?

The basic syntax is:

{{ expression | filterName }}

Example:

{{ product.price | currency }} formats the price as currency.

You can also chain filters like this:

{{ product.name | uppercase | limitTo:10 }}

What happens if you chain multiple filters?

The filters are applied in the order they are written. So if you use {{ value | filter1 | filter2 }}, the output of filter1 becomes the input of filter2. Chaining helps combine multiple formatting options in one expression.

What's the difference between number and currency filters?

- number formats a numeric value with a fixed number of decimal places.
- currency formats the number as money, adding a currency symbol like ₹, $, etc., and also includes decimals by default.

  Example: 1234 | number:2 → 1,234.00 vs 1234 | currency → $1,234.00

Is the filter case-sensitive?

Yes, filters like filter (used for searching) can be case-sensitive depending on how you write them. You can add options to make them case-insensitive or use custom logic in your filter function.

Q2: How did you bind product data to your table?

Main Answer:

I used the ng-repeat directive to bind product data to the table.

Detailed Explanation:

In AngularJS, ng-repeat is used to loop over an array and display each item in a table or list. For example, if you have a list of products in your controller, you can bind each one to a table row like this:

<tr ng-repeat="product in products">

What directive is used for data iteration?
ng-repeat is the main directive used to loop through arrays and bind each item to the DOM.
Can you bind data conditionally?
Yes, using ng-if, ng-show, or conditions inside ng-repeat (like ng-repeat="product in products | filter:{available:true}"). This allows you to display only the data that meets specific conditions.
How do you handle missing values in your table?
You can use conditional expressions like {{ product.name || 'N/A' }} to show a default value if a field is missing or undefined.
What happens when your model changes dynamically?
AngularJS updates the view automatically because of two-way data binding. If the model (data in controller) changes, the table updates in real-time.
What are the performance concerns with large data sets?
With a large number of items, ng-repeat can cause slow rendering or lag because Angular watches every item for changes. To optimize performance, use track by with a unique ID and consider pagination.
Q3: How does orderBy work in AngularJS?
Main Answer:
orderBy is a filter used to sort a list of items in ascending or descending order.
Detailed Explanation:
You apply it inside ng-repeat to sort items by a specific property, such as price or name. Example:
ng-repeat="product in products | orderBy:'price'"
This sorts the list by price in ascending order.
Can you sort by more than one field?
Yes, you can pass an array of field names like this:
orderBy:['category', 'price']
AngularJS will first sort by category, then by price within each category.
How do you sort in descending order?
Use a minus sign (-) before the field:
orderBy:'-price' sorts in descending order.
Can you apply orderBy on filtered data?
Yes, you can combine filters:
ng-repeat="product in products | filter:searchText | orderBy:'price'"
First, it filters the products, then it sorts the result.
What happens when some values are null?
Null values are usually pushed to the end. Sorting may not behave as expected if many values are missing, so it's best to handle such cases manually or provide default values.
What are alternatives if orderBy fails?
You can sort the data manually in the controller using JavaScript's .sort() function and then bind the sorted array to the view.
Q4: What is the purpose of currency and number filters?
Main Answer:
They are used to format numbers for better readability.
Detailed Explanation:

- number filter formats plain numbers to show decimals and comma separators.

- currency filter adds a currency symbol and formats it like money.

These filters help display product prices and values in a professional, user-friendly way.

Can you change the currency symbol?

Yes. You can pass a symbol to the currency filter like this:

currency:'₹' or set a default symbol in the AngularJS configuration.

How to format a number to two decimal places?

Use: {{ price | number:2 }}

This will display two digits after the decimal point.

Are these filters locale-sensitive?

Yes, they can be. AngularJS uses locale settings to determine currency symbols, decimal separators, and formatting style. You can change the locale using Angular's $locale service.

What happens with negative numbers?

They are displayed with a minus sign (like -₹50.00). The formatting remains the same, and the value is shown clearly.

Do they affect data in the model?

No, filters only affect how data is shown in the view. The actual value in the model remains unchanged.

Q5: How is two-way data binding used in this app?

Main Answer:

Two-way data binding keeps the model and view in sync automatically.

Detailed Explanation:

In AngularJS, when a user updates the input field, the model gets updated, and when the model is updated, the view changes too. This is known as two-way binding. It's useful in forms, filters, and real-time UI updates.

What is the directive used for two-way binding?

ng-model is the key directive. It binds the HTML element (like input or select) to a variable in the controller.

Is two-way binding always required?

Not always. For static data, one-way binding is enough. Two-way binding is useful only when the data can change from both the model and the view.

How does AngularJS handle synchronization?

AngularJS uses a "digest cycle" to check for changes in model values and updates the DOM accordingly. This keeps both sides in sync.

What are the downsides of two-way binding?

With large apps or complex UIs, too many bindings can slow down performance due to frequent updates. Also, debugging can become harder if changes come from unexpected places.

How can you convert it to one-way binding?

Use {{ }} (interpolation) or ng-bind for one-way binding from model to view only. Avoid using ng-model if you don't want updates from view to affect the model.

2. Result Management App (React + Node.js + MongoDB)

Q1: What front-end technology did you use? Why?

Main Answer:

I used React.js for the frontend because it's efficient, component-based, and ideal for building dynamic web applications.

Detailed Explanation:

React allows you to break your UI into reusable components, making the code more organized and easier to manage. It uses a virtual DOM which improves performance and allows real-time updates without refreshing the page. I chose React for its strong developer community, support for hooks, and ease of state management.

Why React instead of Angular or Vue?

React is lightweight and has a simpler learning curve compared to Angular. Also, React uses JSX, which feels more like HTML and JavaScript combined, making it intuitive. Vue is good too, but React is more popular in job markets and has more library support.

How does JSX help in UI creation?

JSX lets you write HTML-like code in JavaScript. It's easier to visualize and create components, especially when working with dynamic content.

Did you use any CSS framework?

Yes (if applicable), I used Bootstrap or Material UI for responsive design and pre-styled components, which saved time and effort.

How did you handle component state?

I used React's useState hook to manage input fields and local data. For larger state management, I could use useReducer or context API.

How did you manage routing in React?

I used the react-router-dom library to handle multiple pages like Home, Add Result, and View Results without reloading the page.

Q2: How did you connect Node.js to MongoDB?

Main Answer:

I used Mongoose to connect Node.js with MongoDB, as it simplifies interaction with the database through models and schemas.

Detailed Explanation:

Mongoose is an Object Data Modeling (ODM) library that provides a structure to MongoDB documents. It helps define schemas, perform validations, and handle queries. In Node.js, I connected using a MongoDB URI and managed the connection with mongoose.connect().

What library did you use (e.g., Mongoose)?

I used Mongoose because it provides schema-based solutions and has built-in query functions like find(), save(), updateOne(), and deleteOne().

How did you handle connection errors?

I wrapped the connection in a try-catch block or used .then().catch() to log any connection issues and gracefully handle server startup.

What is the role of the schema?

A schema defines the structure of documents in a collection. For example, it ensures every result has fields like PRN, name, email, and marks.

How do you close the database connection?

Using mongoose.connection.close() in situations like application shutdown or error handling to prevent memory leaks.

What is the use of .env in connection strings?
The .env file is used to store sensitive data like MongoDB connection URIs. It keeps credentials secure and separate from the main code.
Q3: What are the CRUD operations you implemented?
Main Answer:
I implemented Create, Read, Update, and Delete operations to manage result records in the MongoDB database.
Detailed Explanation:

- Create: Add a new student's result with fields like name, PRN, and marks.
- Read: View all results or search for a specific one.
- Update: Edit a student's marks or details.
- Delete: Remove a result from the database.
  These operations were triggered via React forms and handled by backend API routes using Express.

How do you create a new document?
By sending a POST request from the frontend and using new ResultModel(req.body).save() on the backend.
How is update different from replace?
Update modifies specific fields using $set, while replace substitutes the entire document with a new one.
How do you prevent duplicate entries?
By checking if a document with the same PRN exists before inserting or using unique constraints in the schema.
How do you perform partial updates?
By using updateOne() with only the fields that need to be changed.
What happens if the document is not found?
The update/delete operations return a result showing matchedCount: 0, and I handle this with error messages on the frontend.
Q4: How is data stored in MongoDB? What does a result document contain?
Main Answer:
Data is stored as JSON-like documents in MongoDB collections. Each student result is stored as a document with key-value pairs.
Detailed Explanation:
Each document represents a single result and contains fields like:

- class
- PRN
- name
- email
- marks (as an object or array for subjects like DS, DBMS, etc.)
  MongoDB is flexible, so the structure is schema-less by default, but with Mongoose we define a fixed structure.

What datatype is used for marks?
Usually, a number or an object (if subject-wise). Example: marks: { DBMS: 85, DS: 90 }
How are arrays used in your document?

If needed, I can store subject marks as an array of objects like:
marks: [{ subject: "DBMS", score: 85 }, { subject: "DS", score: 90 }]
Can nested documents be created?
Yes, MongoDB supports nested objects. For example, you can have student.details.email.
How do you enforce constraints in MongoDB?
With Mongoose, we define validation rules like required, min, max, and unique in the schema.
What is the _id field?
MongoDB automatically adds an _id field as a unique identifier for each document.
Q5: How is form data sent to the server?
Main Answer:
Form data is sent from the React frontend to the Node.js server using HTTP POST requests, typically with the fetch API or Axios.
Detailed Explanation:
The form collects inputs like name, PRN, marks, etc. On submission, the data is packaged as JSON and sent to the server endpoint (e.g., /api/addResult) using a POST request. The server then parses the data and stores it in MongoDB.
Which HTTP method did you use?
I used POST to add data, GET to fetch data, PUT for updating, and DELETE for removing records.
Did you use Axios or fetch?
I used either based on the project—fetch is built-in and simple; Axios provides easier syntax and better error handling.
How is data encoded during transmission?
Data is sent as JSON with the header Content-Type: application/json.
How did you handle JSON parsing?
The backend uses express.json() middleware to automatically parse incoming JSON data in the request body.
How do you validate form data before sending?
I added frontend validation using HTML attributes or React logic, and also validated it again on the server side using schema validation to ensure safety.

3. Calculator App using TypeScript

Q1: Why did you choose TypeScript for a calculator?
Main Answer:
I chose TypeScript because it adds type safety, better structure, and early error detection during development.
Detailed Explanation:
TypeScript is a superset of JavaScript that supports static typing. In a calculator app, where multiple operations and user inputs are involved, TypeScript helps prevent bugs by ensuring that values passed to functions are of the correct type (e.g., numbers instead of strings). It also improves code readability and makes debugging easier.
What is the advantage of type safety?
It catches errors at compile time instead of runtime. For example, trying to add a number to a string will give an error before running the app.

Can JavaScript do the same?
JavaScript can perform the same logic, but without type checking, errors may go unnoticed until runtime. TypeScript makes the development more reliable and scalable.
How does TypeScript improve code quality?
By forcing the developer to define types, it reduces unexpected behavior and helps tools like VS Code provide better autocomplete and error hints.
What are TS compilation errors you encountered?
Examples include assigning string to a number variable or calling a function with wrong argument types. The compiler highlights such issues before the code runs.
Is it possible to convert your code back to JavaScript?
Yes, TypeScript code is compiled to plain JavaScript using the TypeScript compiler (tsc). The output runs on any browser or JavaScript engine.
Q2: How did you implement functions for arithmetic operations?
Main Answer:
I implemented separate functions for each operation like add, subtract, multiply, and divide, taking input numbers and returning results.
Detailed Explanation:
Each function receives two parameters of type number and returns the computed value. For division, I handled the case where the second number is zero to prevent errors. This modular approach keeps the code clean and reusable.
Did you use arrow functions or normal ones?
I used arrow functions because they are shorter and bind the this keyword lexically, which is helpful inside classes or callbacks.
How do you handle divide-by-zero?
I added a condition to check if the second number is zero before performing division. If true, I return a message like “Cannot divide by zero”.
Can you extend the calculator to support percentages?
Yes, we can easily add more functions like getPercentage(a, b) which returns (a / b) * 100. TypeScript makes it easy to scale.
Did you return values or just display them?
I returned values from the functions and then displayed them in the UI, separating the logic from the presentation.
How do you handle decimal operations?
I use JavaScript’s built-in math operations and format the result using .toFixed(2) if needed. TypeScript ensures both inputs are numbers.
Q3: What TypeScript features did you use?
Main Answer:
I used type annotations, functions, interfaces (if needed), and strict type checking to build the calculator.
Detailed Explanation:

- Type annotations: I declared variable types like let num1: number;
- Functions: Defined reusable blocks for each operation.
- Optional types: Used to handle optional inputs or conditions.

These features helped catch errors early and make the code more readable and predictable.
What is type inference?

TypeScript can automatically detect the type of a variable based on its assigned value. For example, let x = 5 is automatically treated as number.
Did you use enums or tuples?
Not in this basic calculator. Enums or tuples could be used if there were multiple operation types or grouped values.
What is the purpose of interfaces?
Interfaces define a structure for objects. If the calculator had settings or history features, interfaces could enforce consistency.
Did you use union types or generics?
In this app, not necessary, but union types like number | string are useful if inputs can vary. Generics are helpful for reusable logic.
How did you define input/output types?
In every function, I defined input parameters and return type explicitly:
Example – function add(a: number, b: number): number
Q4: Did you use classes? If yes, explain their role.
Main Answer:
Yes, I used a class to group all calculator methods logically under one structure.
Detailed Explanation:
A class like Calculator contains methods like add, subtract, etc. It helps organize the code and makes it reusable. You can create an object of the class and call methods easily. This is helpful if you want to extend features like memory storage, history tracking, or custom operations.
What was defined in the constructor?
If needed, we can initialize default values like result = 0 or accept parameters like the initial value.
What methods were part of the class?
Methods like add(a, b), subtract(a, b), multiply(a, b), divide(a, b) were included. These can return results or update a class-level variable.
How did you instantiate your class?
Using: let calc = new Calculator();
Then called: calc.add(10, 5);
Can you implement private fields?
Yes, using the private keyword to restrict access to internal properties.
What happens if you omit the this keyword?
You won't be accessing the class's own property or method, which can lead to undefined behavior or errors.

4. Single Page App (SPA) using AngularJS

Q1: What is a SPA (Single Page Application)?
Main Answer:
A Single Page Application (SPA) is a web app that loads a single HTML page and dynamically updates content without refreshing the entire page.
Detailed Explanation:
In a SPA, only parts of the page update when the user navigates or interacts with the app. This is done using JavaScript frameworks like AngularJS, which handle routing and data rendering on the client side.

SPAs feel faster and smoother because there's no full-page reload, and only required data is fetched asynchronously.

How does SPA differ from MPA?

MPAs (Multi Page Applications) reload the whole page on every click, while SPAs update the content dynamically without reloading the page.

What are the pros and cons?

Pros: Faster navigation, better user experience, and reduced server load.

Cons: SEO challenges, initial load time can be larger, and requires JavaScript to function.

Does SPA affect SEO?

Yes, since content is loaded dynamically, it's harder for search engines to index. This is a common limitation of SPAs without server-side rendering.

Can SPAs work without JavaScript?

No, SPAs depend heavily on JavaScript for routing and content updates.

What is the role of client-side routing?

It helps change views within the same page without reloading. AngularJS handles this using routes defined in the frontend.

Q2: How does AngularJS support routing?

Main Answer:

AngularJS supports routing using the ngRoute module and $routeProvider service to define paths and associate them with views and controllers.

Detailed Explanation:

With routing, we can load different content or templates based on the URL path. For example, when a user navigates to /home or /about, different partial HTML templates are loaded inside the main page dynamically, keeping the application as a single page.

What module is required for routing?

ngRoute is the AngularJS module required for basic routing. It must be included in the app's dependencies.

How do you define routes?

Using $routeProvider.when() inside the app configuration block. Each route is linked to a template and controller.

Example:

```
$routeProvider.when('/home', { templateUrl: 'home.html', controller: 'HomeCtrl' })
```

Can you add route parameters?

Yes, you can use :paramName in the route URL and access it using $routeParams in the controller.

What is the fallback route?

The fallback route is defined using .otherwise(), which shows a default page if the URL doesn't match any route.

How do you prevent undefined routes?

Using .otherwise({ redirectTo: '/home' }) ensures users are redirected to a default view instead of seeing a blank or error page.

Q3: What is ng-view and how is it used?

Main Answer:

ng-view is a directive that acts as a placeholder where the routed content (templates) will be injected.

Detailed Explanation:
It is placed in the main HTML page (usually index.html) and serves as a container for displaying views loaded through AngularJS routes. When a route is activated, Angular loads the corresponding HTML template into the ng-view area.
How does it interact with $routeProvider?
$routeProvider tells Angular which template to load into ng-view based on the current route.
Can multiple ng-views exist?
No, typically only one ng-view is allowed. For complex routing, you can use ui-router which supports multiple views.
What are the alternatives to ng-view?
ui-view from the ui-router module is a more powerful alternative that supports nested views and state-based routing.
Where should ng-view be placed?
Usually inside the <body> tag in index.html, in the section where you want the content to change.
How do templates work in ng-view?
Templates are partial HTML files that load dynamically into ng-view based on the current route.
Q4: How do SPAs improve performance?
Main Answer:
SPAs improve performance by loading content dynamically without reloading the full page, which makes the app feel faster and smoother.
Detailed Explanation:
After the initial page load, only the data or view changes based on user actions. No full HTML reload is needed. This reduces bandwidth usage and server load. Also, transitions between views are quicker because only a part of the page changes.
What is lazy loading?
Lazy loading is a technique where parts of the application (like components or templates) are loaded only when required. This speeds up the initial load and improves performance.
How does caching help in SPAs?
Once resources like JavaScript files and templates are loaded, they are cached in the browser, so repeated visits are faster.
Does SPA reduce server load?
Yes. Since only data is fetched using APIs and the main page remains static, fewer full-page requests hit the server.
How are assets handled in SPAs?
Assets like CSS, JavaScript, and templates are typically loaded once and reused throughout the session.
Can SPAs work offline?
With the help of Service Workers (in more advanced setups), SPAs can work offline by caching data and pages locally.

5. AngularJS Web Services App

Q1: What is a web service?

Main Answer:
A web service is a way for two applications to communicate over a network using standard formats like HTTP and JSON or XML.
Detailed Explanation:
In simple terms, a web service allows a frontend (like AngularJS) to get or send data to a backend server or API. It works over the internet using HTTP methods like GET, POST, PUT, and DELETE. For example, when we want to fetch user data or submit a form, we use web services to communicate with the backend.
How is REST different from SOAP?
REST is lightweight, uses JSON, and works with standard HTTP methods. SOAP is more complex, uses XML, and has strict rules.
What are JSON and XML used for?
They are data formats used to send and receive structured information between client and server.
Can you build your own API?
Yes, using backend technologies like Node.js, Python (Flask/Django), or PHP, we can build custom APIs that send or receive data.
What is an endpoint?
An endpoint is a specific URL where the frontend sends a request to access or modify data, like /api/users.
What is a typical web service response?
Usually a JSON object with data (like { name: "John", age: 25 }) or a status message like { success: true, message: "Data saved" }.
Q2: How does AngularJS communicate with the backend?
Main Answer:
AngularJS communicates with the backend using the $http service, which allows sending asynchronous HTTP requests to APIs or servers.
Detailed Explanation:
With $http, we can make GET, POST, PUT, and DELETE requests to fetch or update data from a web service. The $http service returns a promise, allowing us to handle responses or errors using .then() and .catch() methods.
What is $http.get used for?
To fetch data from the backend, like getting a list of users, products, or any database records.
How are headers added?
Headers like Content-Type or authorization tokens can be added in the $http config object during the request.
How do you send data with POST?
Using $http.post(url, dataObject), where dataObject contains the data to be submitted to the server.
How do you handle CORS?
CORS (Cross-Origin Resource Sharing) is handled on the backend by allowing specific domains to access the API using headers like Access-Control-Allow-Origin.
What is the $q service?
$q is used to create and manage promises manually, giving more control over asynchronous behavior in AngularJS.
Q3: Did you use $http or $resource for API calls?

Main Answer:
I used $http because it's simpler and more flexible for basic API requests.
Detailed Explanation:
$http is straightforward and allows you to send any type of HTTP request manually. $resource is part of ngResource and is more suitable for RESTful services where you deal with structured resources and models. For learning or simple apps, $http is easier to manage.
What is the main difference?
$http gives full control over the request, while $resource is more model-based and designed for CRUD operations.
Which one is preferred for RESTful APIs?
$resource is often used for RESTful services, especially when working with structured objects and fewer customizations.
Can you create custom methods in $resource?
Yes, $resource allows you to define custom methods like update, save, or delete with specific configurations.
What is the default behavior of $http.post?
It sends data in the request body to the backend, usually as JSON.
Why might $http be easier to debug?
It's more transparent—you can see the full request and response and handle everything explicitly with .then() or .catch().
Q4: How do you handle success and error responses in AngularJS?
Main Answer:
We handle responses using promises, specifically .then() for success and .catch() or .error() for failures.
Detailed Explanation:
When an HTTP request is made using $http, AngularJS returns a promise. In .then(), we can access the data returned from the server. In .catch() or .error(), we can handle errors like failed connections, 404 (Not Found), or 500 (Server Error).
What is .then() and .catch()?
.then() runs when the request is successful. .catch() runs when there is an error. These help us handle both outcomes gracefully.
Can you retry failed requests?
Yes, you can add retry logic inside .catch() or create a retry function that calls the same API again if needed.
How do you log errors?
Using console.error() or showing a user-friendly message using alert() or displaying it on the UI.
What happens on 404 and 500 errors?

- 404: The requested API endpoint is not found.
- 500: Server-side error due to code failure or database issues.
  These are caught in the .catch() block.

How do you show error messages to the user?
By updating a variable like errorMsg in the controller and showing it in the view using ng-if or ng-show.

6. AJAX-Based Web Application

Q1: What is AJAX and why is it used?
Main Answer:
AJAX stands for *Asynchronous JavaScript and XML*. It is used to update parts of a web page without reloading the entire page.
Detailed Explanation:
AJAX allows communication with a server in the background using JavaScript, so the user experiences smooth, real-time interactions. For example, when you search in Google and results update while typing—that's AJAX in action. It improves performance and user experience by avoiding full-page reloads and reducing data transfer.
What are the main advantages of AJAX?

- Faster updates
- Reduced server load
- Better user experience
- Real-time data fetching

Does AJAX work without JavaScript?
No. AJAX depends completely on JavaScript to send and receive data.
What data formats does AJAX support?
Primarily JSON and XML, but it also supports plain text and HTML.
Is AJAX synchronous or asynchronous by default?
It is asynchronous by default, but it can be configured to run synchronously (not recommended as it blocks the UI).
How does AJAX differ from traditional page requests?
Traditional requests reload the whole page. AJAX only updates specific parts by sending data in the background.
Q2: How did you implement asynchronous behavior?
Main Answer:
I used AJAX calls with either XMLHttpRequest or the fetch API to send requests without blocking the page.
Detailed Explanation:
When a user performs an action (like typing in a search box), I send an AJAX request to the backend using JavaScript. While the request is processed, the user can continue interacting with the page. When the response is received, the relevant part of the page updates dynamically.
What is the role of callbacks?
Callbacks handle the response once the server replies, allowing you to process data or update the UI.
Did you use async/await or promises?
Yes, async/await or .then() methods with fetch were used to simplify asynchronous code and improve readability.
How do you handle execution order?

Asynchronous functions run in the background. The .then() block ensures that the code inside it runs after the server responds.
What if the server takes too long to respond?
A timeout can be added, and a loading indicator is usually shown to inform the user that the system is processing.
How do you cancel a request?
Modern APIs like AbortController (with fetch) can be used to cancel pending AJAX requests.
Q3: What is the role of XMLHttpRequest or fetch API in your project?
Main Answer:
I used fetch API (or XMLHttpRequest) to send asynchronous HTTP requests and receive responses from the server.
Detailed Explanation:

- XMLHttpRequest is the traditional method for AJAX.
- fetch is a modern, simpler alternative that supports promises.
  Both are used to perform tasks like sending form data, fetching product lists, or updating the UI dynamically without reloading.

Which one did you use and why?
I used fetch because it's easier to write, uses promises, and supports async/await, which makes the code more readable.
How do you handle errors in fetch?
Using .catch() or try-catch blocks with async/await. Also, I check the HTTP status code (e.g., 200 for success).
What are the default request methods?
GET is the default method for both fetch and XMLHttpRequest.
How do you send headers with fetch?
Headers can be passed as part of the request options:
fetch(url, { method: 'POST', headers: { 'Content-Type': 'application/json' } })
Can you upload files using fetch?
Yes, using FormData and setting the request body accordingly.
Q4: How did you handle the response without reloading the page?
Main Answer:
I used JavaScript DOM manipulation to update the content on the page with the response data.
Detailed Explanation:
Once the server sends a response (like a product list or search result), I used innerHTML, appendChild(), or other DOM methods to display the data in a specific section of the page. This happens without refreshing the entire page, which creates a smoother user experience.
What DOM methods did you use?
Commonly used methods include:

- document.getElementById()
- .innerHTML = responseData
- createElement() and appendChild()

How did you parse the JSON?
Using the .json() method with fetch, or JSON.parse() if using XMLHttpRequest.
How did you handle dynamic content updates?

Cleared the old content (if needed) and inserted new HTML based on the response. Used loops for lists.
What happens if the response is malformed?
The app may throw a JSON parsing error. I handled this using try-catch or .catch() to show a friendly error message.
How do you clear previous results?
Set the target element's .innerHTML = '' before appending new data.

7. MongoDB CRUD Operations

Q1: How did you install and set up MongoDB?
Main Answer:
I installed MongoDB from the official website and set it up locally using the MongoDB Community Server and MongoDB Compass for GUI operations.
Detailed Explanation:
After downloading the MongoDB installer, I followed the setup instructions, which included installing MongoDB as a service. Then I used mongod to start the server and mongo to open the shell. I also used MongoDB Compass to create and manage databases visually. In Node.js projects, I connected using the Mongoose library.
What command starts the MongoDB service?
On Windows: net start MongoDB or just run mongod
On Linux/macOS: Use sudo systemctl start mongod
What is mongod and mongo?

- mongod: Starts the MongoDB database server
- mongo: Opens the MongoDB shell to interact with the server

Did you use Compass or CLI?
I used both. Compass is good for visual management, while CLI is useful for quick queries and scripting.
How do you check the status of MongoDB?
Using: mongo shell and running db.stats() or use dbName
What is a MongoDB URI?
A URI is the connection string used by applications to connect to MongoDB.
Example: mongodb://localhost:27017/myDatabase
Q2: What commands are used for basic CRUD in MongoDB?
Main Answer:
CRUD stands for Create, Read, Update, and Delete. MongoDB supports these using specific commands in the shell or through Mongoose in code.
Detailed Explanation:

- Create → insertOne() or insertMany()
- Read → find() or findOne()
- Update → updateOne(), updateMany()
- Delete → deleteOne(), deleteMany()

These operations allow us to manage records (documents) in MongoDB collections.
What is the syntax for find()?
db.collection.find({ field: value }) returns matching documents.
How do you insert a document?

Using: db.collection.insertOne({ name: "John", age: 25 })
What operator is used in updateOne()?
The $set operator is commonly used to update specific fields.
Example: { $set: { age: 26 } }
How do you delete multiple documents?
Using deleteMany({ field: value }) to remove all matching documents.
What is upsert?
It's an option in update operations that inserts a new document if no match is found.
Q3: What is the difference between insertOne() and insertMany()?
Main Answer:
insertOne() adds a single document to the collection, while insertMany() adds multiple documents at once.
Detailed Explanation:
Both are used to insert data into MongoDB.

- Use insertOne() when adding just one record.
- Use insertMany() when you want to insert an array of documents in one go, which is more efficient and faster.

What happens if one document in insertMany() fails?
If ordered: true (default), the operation stops at the first error. If ordered: false, it continues inserting the remaining documents.
Which one is faster?
insertMany() is faster for bulk inserts because it reduces the number of database operations.
What is the return value of each?

- insertOne() returns an object with acknowledged: true and the _id of the inserted document.
- insertMany() returns an array of _ids.

Can you insert empty arrays?
No, inserting an empty array will throw an error or be ignored. You must insert valid documents.
Q4: How do you update or delete a specific document?
Main Answer:
We use updateOne() or deleteOne() with a condition that matches the document we want to change or remove.
Detailed Explanation:
To target a specific document, we use a filter like { _id: ObjectId("...") } or any unique field.

- To update: updateOne({ name: "John" }, { $set: { age: 30 } })
- To delete: deleteOne({ name: "John" })
  Mongoose also provides functions like .findByIdAndUpdate() in Node.js apps.

How do you target by _id?
Using: { _id: ObjectId("id_here") }
Make sure to import ObjectId if using it in code.
What does $set do?
$set updates only the specified fields without modifying the rest of the document.
Can you use regex in queries?
Yes, MongoDB supports regular expressions in find() queries.
Example: { name: /^J/ } finds names starting with "J".

How do you confirm deletion?
Check the result returned by deleteOne(), which shows deletedCount. If it's 1, the document was successfully deleted.

8. AngularJS Registration Form with Validation

Q1: What is form validation in AngularJS?
Main Answer:
Form validation in AngularJS ensures that user input meets specific criteria before the form is submitted.
Detailed Explanation:
AngularJS provides built-in directives for client-side validation, such as ng-required, ng-pattern, and ng-minlength. These help to check whether the form fields are correctly filled out—like checking if an email is valid or a password is long enough. Angular automatically updates the form's validity state and provides classes like ng-valid or ng-invalid.
What are the types of validation?

- Required field validation
- Pattern matching (e.g., email format)
- Minimum/maximum length
- Number range checks

How does AngularJS track form state?
Using built-in properties like $valid, $invalid, $touched, $dirty, and $pristine on form and input elements.
What is $dirty and $touched?

- $dirty: True if the input value has been changed.
- $touched: True if the input has been focused and then blurred.

How do you reset a form?
You can reset form values and validity using Angular methods in the controller, or by reinitializing the model bound with ng-model.
Can validations be customized?
Yes, AngularJS allows creation of custom validation directives for specific use cases not covered by default validations.
Q2: What are ng-required, ng-pattern, and ng-minlength?
Main Answer:
These are AngularJS directives used to apply validation rules directly to form inputs.
Detailed Explanation:

- ng-required: Marks a field as mandatory.
- ng-pattern: Validates input based on a regular expression.
- ng-minlength: Ensures the input is not shorter than the given number of characters.
  They work together with ng-model to bind and validate form data.

What kind of regex did you use?
For email: /^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$/
For password: /^[0-9]{6}$/ to ensure exactly 6 digits.
Can you apply multiple patterns?
You can use combined conditions in a single regex or stack multiple validation rules using AngularJS.

What's the difference between required and minlength?

- required checks if the field is filled.
- minlength checks how many characters have been entered.

How do these affect form submission?

If these validations fail, the form will not be considered valid and the submit button can be disabled using ng-disabled.

Q3: How did you show error messages conditionally?

Main Answer:

I used ng-show or ng-if along with AngularJS form validation state properties to show error messages only when needed.

Detailed Explanation:

For each input field, I checked if it was invalid and touched (meaning the user interacted with it). Then I displayed the error message accordingly. Example:

Email is required

What directive did you use?

I mainly used ng-show, ng-if, and sometimes ng-messages for grouped error handling.

Can errors be shown only after the field is touched?

Yes. By checking $touched property, we avoid showing errors before the user interacts with the input.

What is the role of ng-if vs ng-show?

- ng-show: Hides or shows the element but keeps it in the DOM.
- ng-if: Completely removes or adds the element in the DOM based on condition.

How do you style error messages?

By applying CSS classes like .error or using Angular's ng-invalid class to highlight fields in red.

Q4: Why did you disable the submit button?

Main Answer:

To prevent form submission until all validations are satisfied and the form is valid.

Detailed Explanation:

Using ng-disabled="formName.$invalid", we ensure that the user cannot submit the form if any required fields are missing or invalid. This provides a better user experience and prevents invalid data from reaching the backend.

What directive is used?

ng-disabled is used on the <button> element to enable or disable submission.

How do you check overall form validity?

By using formName.$valid or formName.$invalid properties which track the entire form state.

Can the user force submit the form?

Not easily from the UI. But it's important to also validate on the server-side to protect against manual or malicious submissions.

How does AngularJS handle novalidate?

The novalidate attribute disables native HTML5 validation, allowing AngularJS to take full control of form validation.

9. Employee Salary Utility in TypeScript

Q1: What is an arrow function in TypeScript?
Main Answer:
An arrow function is a shorter way to write a function in TypeScript, introduced in ES6, with cleaner syntax and lexical this binding.
Detailed Explanation:
Arrow functions use the => syntax and are often used in callbacks or inline function expressions. They are useful because they do not redefine the value of this inside their body, which helps avoid common bugs in object-oriented programming.
Example:
const add = (a: number, b: number): number => a + b;

What is the syntax?
(parameters) => { statements }
Or for a one-liner: (a, b) => a + b
How is this handled differently?
In arrow functions, this refers to the surrounding context (lexical scope). It does not get its own this, unlike traditional functions.
Can arrow functions have default parameters?
Yes. You can define defaults like:
(a: number, b: number = 10) => a + b
Are they anonymous?
Yes, unless assigned to a variable or used inside an object.
Can you return objects from arrow functions?
Yes. Wrap the object in parentheses:
() => ({ name: "John" })
Q2: How did you calculate bonuses?
Main Answer:
I calculated bonuses based on each employee's performance rating using conditional logic inside arrow functions.
Detailed Explanation:
Each employee had a baseSalary and a performanceRating. Based on the rating, I used logic to calculate a bonus percentage and added it to the base salary. For example, if the rating was 10, a 20% bonus was given; if it was 5, a 10% bonus. This final amount was returned and displayed for each employee.
What is your logic based on ratings?

- Rating 10 → 20% bonus
- Rating 5 → 10% bonus
- Others → no bonus or fixed amount

How do you prevent negative values?
By validating the salary and rating fields to ensure they are positive before doing calculations.
Did you use conditionals or switch cases?
I used if-else or ternary operators inside arrow functions for cleaner syntax.
Can the bonus be a percentage or flat rate?
Yes, TypeScript supports both approaches. It depends on business logic.

Q3: What logic did you use for performance ratings?
Main Answer:
I used conditional checks to assign bonus percentages based on performance ratings like 5 or 10.
Detailed Explanation:
The performance rating was evaluated using if-else logic. For example, if an employee had a rating of 10, they received a 20% bonus. This logic was implemented inside a reusable arrow function.
What were the possible ratings?
Typically, 5 and 10 were used, but you can extend this for other values like 7 or 8.
How do you validate the rating input?
By checking if the value is a number and lies within the acceptable range (e.g., 0 to 10).
How is the final salary computed?
finalSalary = baseSalary + (baseSalary * bonusPercentage)
Can ratings affect other parameters?
Yes, ratings can also be used to assign promotions or performance messages in the future.
Q4: How did you structure your employee array?
Main Answer:
I structured the employees as an array of objects, where each object represents an employee with their name, salary, and performance rating.
Detailed Explanation:
Each object contained fields like:
{ name: "Alice", baseSalary: 50000, performanceRating: 10 }
This array was iterated using map() to calculate the final salary with bonuses for each employee.
What properties does each object have?

- name: string
- baseSalary: number
- performanceRating: number

How did you access individual employees?
Using forEach or map loops:
employees.map(emp => emp.name)
Did you use array methods like map() or forEach()?
Yes. map() was used to return a new array with calculated final salaries.
How do you search for a specific employee?
Using find() method:
employees.find(emp => emp.name === "John")

10. Appointment Booking System (MongoDB + REST API)

Q1: What is a RESTful API?
Main Answer:
A RESTful API is an interface that allows applications to communicate with each other over HTTP using standard methods like GET, POST, PUT, and DELETE.
Detailed Explanation:
REST stands for *Representational State Transfer*. It follows a set of rules and principles to design

scalable and stateless APIs. Each resource (like a doctor or appointment) is represented by a unique URL (endpoint), and different HTTP methods perform actions on them:

- GET: retrieve data
- POST: create new data
- PUT: update data
- DELETE: remove data

REST APIs are widely used in web development to connect frontend apps (like Angular, React) with backend services (like Node.js with MongoDB).

What are the main HTTP methods?

- GET → Read data
- POST → Create new resource
- PUT/PATCH → Update existing resource
- DELETE → Remove resource

What does stateless mean in REST?

Each request is independent; the server does not remember previous requests. This makes REST APIs scalable and fast.

What is an endpoint?

An endpoint is a URL path that handles a specific operation, like /api/appointments or /api/doctors.

How do you structure URL paths?

Using resource-based paths:

/api/doctors → list all doctors

/api/appointments/:id → get or update a specific appointment

Q2: How did you implement booking and canceling appointments?

Main Answer:

I used POST requests to book new appointments and DELETE or PUT requests to cancel or update existing ones.

Detailed Explanation:

- When a user books an appointment, a POST request is sent with details like doctor name, date, and time, and stored in a MongoDB collection.
- For cancellation, either a DELETE request is sent with the appointment ID or a PUT request is made to mark the status as “cancelled”.

This ensures all appointment data is stored and updated properly in the backend.

What collections were affected?

The appointments collection stores booking details, and optionally, a doctors collection stores doctor profiles.

How do you prevent double bookings?

By checking in the database whether the selected doctor already has a booking for that date and time before saving the new appointment.

What fields are needed for a booking?

Common fields include:

- Patient name
- Doctor name
- Date
- Time

- Status (e.g., booked, cancelled)

How is cancellation logged or handled?

We can either delete the appointment or update its status field to “cancelled” so it stays in records.

Q3: How did you structure your MongoDB collections?

Main Answer:

I created collections like doctors and appointments, where each document represents an individual record with specific fields.

Detailed Explanation:

- doctors: contains name, specialization, availability, and ID.
- appointments: contains patient info, doctor ID (as reference), date, time, and status.

  Documents were stored in JSON format and linked logically, such as referencing doctor IDs in appointments.

What data types were used?

- Strings for names
- Dates for appointment times
- Boolean or enums for status
- ObjectId for referencing related documents

Were there any nested documents?

If needed, I could nest patient details inside the appointment object, but generally I kept related data separate and used referencing.

How do you handle references between collections?

Using the doctor’s _id as a foreign key in the appointments collection.

What indexes did you define?

Indexes can be added on fields like doctorId or date to speed up search and prevent duplicate bookings.

Q4: What HTTP methods did you use?

Main Answer:

I used standard REST methods:

- GET to fetch doctors and appointments
- POST to book new appointments
- PUT to update them
- DELETE to cancel

Detailed Explanation:

Each method mapped to a different action:

- GET /api/doctors → Fetch all doctors
- POST /api/appointments → Add a new appointment
- PUT /api/appointments/:id → Edit details
- DELETE /api/appointments/:id → Remove or cancel an appointment

  All requests were handled in Express.js and connected to MongoDB via Mongoose.

What is the difference between PUT and PATCH?

- PUT replaces the whole document.
- PATCH updates only the specified fields.

Which method is used for delete?

DELETE method. Example: DELETE /api/appointments/123

How do you send data in POST requests?

In JSON format in the body of the request.
Use Content-Type: application/json in headers.
What is the response structure?
Usually, a JSON response with a success message or the inserted/updated data.
Example: { success: true, message: "Appointment booked" }

11. AJAX Product Catalog Search

Q1: How is dynamic search implemented using AJAX?
Main Answer:
Dynamic search using AJAX is implemented by sending requests to the server as the user types, and updating the product list in real time without reloading the page.
Detailed Explanation:
An event listener (like onkeyup) detects when the user types in the search box. This triggers an AJAX request (using XMLHttpRequest or fetch) to the server with the search query. The server returns filtered results, which are then displayed on the web page immediately.
What is the trigger event?
Usually onkeyup or input events on the search box are used to detect typing.
How is the query string formed?
By appending the search text to the URL, like /search?query=mobile.
Did you debounce the search input?
Debouncing is used to wait for a pause in typing before sending the request, reducing unnecessary calls. It improves performance.
How do you handle no matches?
If the server returns an empty array, I display a message like “No results found.”
Q2: How do you filter data from the backend?
Main Answer:
Data is filtered on the server side by matching the search query against fields like product name or category using regular expressions or string matching.
Detailed Explanation:
In the backend (Node.js, Python, etc.), we take the query from the request and use it to filter the product collection. For MongoDB, we can use $regex to match partial strings, making the search more flexible.
What query parameters are used?
Commonly ?search=term or ?q=term, sent through GET requests.
Did you use regular expressions?
Yes, to allow partial and case-insensitive matching.
Example: { name: { $regex: searchTerm, $options: 'i' } }
How do you secure the API against injections?
By validating and sanitizing inputs before using them in database queries.
Is filtering case-sensitive?
By default, yes. But adding the i option to regex makes it case-insensitive.
Q3: Did you use onkeyup or event listeners?
Main Answer:
Yes, I used an event listener for keyup on the search input to detect real-time user input and trigger the AJAX call.

Detailed Explanation:
The keyup event ensures the value is updated after each key press. I attached it using addEventListener() in JavaScript and called a function that fetches matching data from the server.
What is the difference between keyup, keydown, and input?

- keydown: Fires when a key is pressed
- keyup: Fires when a key is released
- input: Fires whenever the input value changes (more reliable for search)

Did you attach listeners dynamically?
Yes, the listener was added when the page loaded, targeting the search input field.
How do you remove event listeners?
Using removeEventListener() and passing the same function reference.
How do you prevent too many API calls?
By using debouncing, which waits for a pause in typing before sending the request.
Q4: How is the UI updated in real-time?
Main Answer:
The UI is updated using JavaScript DOM manipulation based on the server response, without reloading the page.
Detailed Explanation:
Once the AJAX response is received, I parse the JSON and create or update HTML elements dynamically using innerHTML, createElement(), or similar methods. This gives a smooth experience where the product list updates instantly as the user types.
What DOM methods were used?

- getElementById()
- innerHTML
- appendChild()
- createElement()

How do you clear old results?
Before showing new data, I set the container's innerHTML to an empty string to remove previous results.
How is the user notified of search progress?
Optionally, a "Loading..." message or spinner is shown while the data is being fetched.
How do you handle loading indicators?
Using CSS to show/hide a loading animation when the request starts and ends (.style.display = "block" or "none").

12. AngularJS Registration Form with Password Pattern

Q1: What validation pattern did you use for the password?
Main Answer:
I used the ng-pattern directive with a regular expression to ensure the password is exactly 6 digits, as required.
Detailed Explanation:
The regex I used was /^[0-9]{6}$/, which checks that the password input has only 6 numeric characters. This ensures consistent formatting and avoids weak or invalid passwords. AngularJS automatically marks the input as valid or invalid based on whether it matches the pattern.
Why exactly 6 digits?

It was specified in the requirement—to accept passwords that are numeric and fixed-length (6 digits).
What regex did you use?
/^[0-9]{6}$/

- ^ and $ ensure the string starts and ends after exactly 6 digits
- [0-9] specifies only digits
- {6} enforces the length

How do you handle leading zeros?
The pattern accepts leading zeros, so values like "012345" are valid.
Is the validation case-sensitive?
No, because the pattern only allows digits; case sensitivity applies to letters, not numbers.
Q2: What is ng-submit and how does it work?
Main Answer:
ng-submit is an AngularJS directive used to call a function when the form is submitted, usually for sending or processing form data.
Detailed Explanation:
When the user clicks the submit button or presses Enter, AngularJS triggers the function defined in ng-submit. This function usually checks if the form is valid (formName.$valid) and then sends the data or performs some logic. Unlike the normal HTML form submit, ng-submit prevents the default page reload and gives full control to the AngularJS controller.
How is it different from normal form submission?
It avoids a page reload and instead triggers a function inside the AngularJS controller.
Can you prevent form refresh?
Yes, ng-submit automatically prevents default behavior. If using onsubmit, you'd have to call event.preventDefault() manually.
How do you bind ng-submit to a function?
In the form tag:
<form name="regForm" ng-submit="submitForm()">

What happens if the form is invalid?
You can prevent submission inside the function by checking regForm.$valid or disable the button using ng-disabled.
Q3: How is the data submitted and handled?
Main Answer:
The form data is collected using ng-model and passed to the controller function through ng-submit. It can then be sent to a backend server via $http.post.
Detailed Explanation:
Each input field uses ng-model to bind to a scope variable. When the user submits the form, these variables are used inside the controller function (like submitForm()). If connected to a backend, the data is sent via AJAX (usually using $http service). If not, it's processed or validated locally.
What happens after validation?
If the form is valid, the data is processed—saved in local storage, sent to a server, or used to trigger other logic like displaying a success message.
How do you pass data to the controller?

Using ng-model, which creates a two-way binding between the input and a variable in the controller's $scope.
Did you use $http for sending data?
If a backend is present, yes. $http.post('/api/register', $scope.userData) can be used to send the form data.
Is the form reset after submission?
Yes, it can be reset using $scope.userData = {} or calling a form reset method.
Q4: How is the form made user-friendly?
Main Answer:
I used real-time validation, error messages, visual feedback, and disabled submission until the form was valid.
Detailed Explanation:

- Input fields show immediate feedback using ng-show or ng-if to display errors.
- Fields are styled with colors (e.g., red borders for invalid inputs).
- The submit button is disabled using ng-disabled="regForm.$invalid" until all inputs are correctly filled.
  This helps guide the user and prevent mistakes.

Did you use placeholders?
Yes, to show example input formats like "Enter your 6-digit password".
How do you guide users with validation hints?
Below each input, messages like "Password must be 6 digits" appear when validation fails.
How do you visually indicate errors?
By using ng-invalid or applying CSS classes when the field is touched and invalid.
How do you group related fields?
Using <fieldset> and <legend>, or layout with Bootstrap or CSS grid/flexbox for organized sections.

13. Shape Interface in TypeScript

Q1: What is an interface in TypeScript?
Main Answer:
An interface in TypeScript defines a structure or contract that a class or object must follow, without providing actual implementation.
Detailed Explanation:
Interfaces help define the properties and methods that must exist in a class or object. They are mainly used to enforce consistency in the code. For example, if you have different shapes like Circle, Rectangle, and Triangle, all can implement a common interface like Shape which includes a method getArea(). This ensures each shape must define its own way of calculating area.
How is it different from a class?
An interface cannot have implementation, while a class has both structure and logic. Classes can implement interfaces.
Can interfaces contain implementation?
No, interfaces only declare what must exist. They do not define how it works.
Can a class implement multiple interfaces?
Yes, a class can implement more than one interface using a comma-separated list.
Can interfaces extend other interfaces?
Yes. One interface can inherit from another, adding or modifying properties.

Q2: How did you implement polymorphism using getArea()?
Main Answer:
I implemented polymorphism by defining getArea() in the Shape interface, and then each class provided its own unique implementation of that method.
Detailed Explanation:

- The Shape interface declared getArea(): number.
- Each class (Circle, Rectangle, Triangle) implemented the interface and wrote its own version of getArea() according to its formula.
  When using an array of different shapes, I could call getArea() on each object and it would behave according to the specific class—this is runtime polymorphism.

How does getArea() behave differently in each class?
Each class uses a different formula:

- Rectangle: length × breadth
- Circle: π × radius²
- Triangle: 0.5 × base × height

What is the role of method overriding?
Each class overrides the getArea() method from the interface to provide its own logic.
Did you use a common function to call getArea()?
Yes, by storing shape objects in an array of type Shape[] and looping through them to call getArea() polymorphically.
Can polymorphism work without interfaces?
It can work using base classes and inheritance, but interfaces provide a more flexible way to enforce structure across unrelated classes.
Q3: How is the getArea() method different for each shape?
Main Answer:
Each shape class (Rectangle, Circle, Triangle) uses a different formula inside its getArea() method to calculate area.
Detailed Explanation:
Even though the method name getArea() is the same, the internal logic varies:

- Rectangle: length * breadth
- Circle: Math.PI * radius * radius
- Triangle: 0.5 * base * height
  This is a classic example of method overriding, where each class defines its own behavior.

What formula did you use?
For Circle: Area = πr², using Math.PI
For Rectangle: Area = length × breadth
For Triangle: Area = 0.5 × base × height
How do you handle invalid inputs?
Optional checks can be added to ensure values like length, radius, etc. are positive before calculating the area.
Are parameters passed to constructors?
Yes, I used constructors to initialize values like radius, length, base, etc.
How is the return value displayed?
Using console.log(), or returned to another function or frontend element for display.

Q4: Why did you choose interface over abstract class?
Main Answer:
I chose interface because it is simpler, and it allows me to enforce structure without worrying about inheritance or implementation.
Detailed Explanation:

- Interfaces are ideal when you just want to define a contract (like getArea()) without any logic.
- Unlike abstract classes, interfaces support multiple implementations and don't require a class hierarchy.
- They are also more flexible and reusable across unrelated classes.

What limitations do abstract classes have?
You can extend only one abstract class, but a class can implement multiple interfaces. Abstract classes may include implementation, which was not needed here.
Can abstract classes implement logic?
Yes, they can have both abstract methods (to be defined later) and fully implemented methods.
Which is better for multiple shape types?
Interfaces are better when you want different shapes to implement common behavior without forcing them into the same class hierarchy.
Can interfaces have optional properties?
Yes, using ? in TypeScript, like:
interface Person { name: string; age?: number }

14. Inheritance (Single & Multilevel) in TypeScript

Q1: What is single and multilevel inheritance?
Main Answer:
Single inheritance is when one class inherits from another.
Multilevel inheritance is when a class inherits from a class that itself inherits from another class.
Detailed Explanation:
Inheritance is a core concept of object-oriented programming. It allows a class (child) to reuse the properties and methods of another class (parent).

- In single inheritance, a subclass directly inherits from a superclass.
- In multilevel inheritance, a class inherits from a subclass which already inherited from a parent class, forming a chain.

Give a short example of each:

- Single:
  class Dog extends Animal
- Multilevel:
  class Animal, class Dog extends Animal, class Puppy extends Dog

How does the extends keyword work?
It allows one class to inherit from another in TypeScript. It brings in all public and protected properties/methods from the base class.
Can constructors be inherited?
No. Constructors are not inherited, but you can call the parent constructor using super() from the child class.

What happens if the same method is in multiple classes?
The method in the child class overrides the one from the parent class.
Q2: Can you give examples of both types?
Main Answer:
Yes, I implemented both types to demonstrate how inheritance works and how methods and properties can be reused and extended.
Detailed Explanation:

- Single inheritance:
  class Car extends Vehicle – Car inherits all the methods from Vehicle like start() or stop()
- Multilevel inheritance:
  class SportsCar extends Car – Now SportsCar can use both methods from Car and Vehicle, forming a chain of inheritance.

What happens if you call a method from the grandparent class?
If it's not overridden, it runs as-is. If overridden in the child class, the child's version will run.
Can you override a method in the middle of the chain?
Yes. For example, if Car overrides a method from Vehicle, and SportsCar doesn't override it again, it will use Car's version.
How do you use super() in multilevel inheritance?
You use super() to call the immediate parent class's constructor or method, not the grandparent directly.
Q3: How is inheritance useful in real applications?
Main Answer:
Inheritance helps in reusing code, reducing redundancy, and maintaining consistency in real applications.
Detailed Explanation:
Instead of writing the same methods or properties again, you can define them in a base class and reuse them in child classes. For example, in an employee management system, you might have:

- Base class: Employee
- Derived classes: Manager, Intern, etc.
  All employees share common attributes like name, id, salary, which are inherited from the Employee.

How does it improve code reuse?
Once common functionality is written in a base class, it can be reused by all child classes, avoiding duplication.
Can it reduce bugs?
Yes, because common logic is written only once, making it easier to test, debug, and maintain.
What are real-world analogies?

- A vehicle is a base class; a car or bike is a subclass.
- An electronic device is a base class; a phone or laptop extends it.

When should you avoid inheritance?
When classes are not truly related, or when inheritance makes the code too complex. In such cases, composition (using objects within objects) is better.
Q4: Did you use extends in your code?
Main Answer:
Yes, I used the extends keyword in TypeScript to implement both single and multilevel inheritance.

Detailed Explanation:
The extends keyword allowed one class to inherit properties and methods from another. For example:
class Car extends Vehicle { ... }

This means Car gets access to all public and protected members of Vehicle.
Can a class extend multiple classes in TypeScript?
No, TypeScript (like JavaScript) does not support multiple inheritance. A class can extend only one other class.
What does super() do?
super() is used inside the child class constructor to call the parent class constructor. It can also be used to call parent class methods.
How do you override constructors?
By writing a constructor in the child class and using super() to initialize the parent class's part.

15. Vehicle Inheritance with Method Overrides (TypeScript)

Q1: What is method overriding?
Main Answer:
Method overriding is when a subclass provides a new implementation for a method that it inherits from its parent class.
Detailed Explanation:
Overriding allows a child class to change or enhance the behavior of a method that is already defined in the parent class. In TypeScript, both parent and child classes can have a method with the same name, and when the method is called on the child class object, the child's version is executed.
What is the difference from overloading?

- Overriding is redefining a method in a child class.

- Overloading is defining multiple methods with the same name but different parameters (TypeScript supports this partially).

How do you ensure the base method is not called?
By overriding it completely in the child class and not calling super.methodName().
What is runtime polymorphism?
It means the method that runs is decided at runtime, based on the object's actual type—not the reference type. This is achieved through overriding.
Q2: How did you override the method in derived classes?
Main Answer:
I created a base class Vehicle with a method like move(), and then overrode it in subclasses like Car and Bicycle with specific implementations.
Detailed Explanation:
Each subclass redefined the move() method to show different outputs. For example, the Car class could print "Car drives on roads", while Bicycle could print "Bicycle is pedaled by user". This shows how the same method behaves differently depending on the object type.
What method did you override?
I overrode a method like describe() or move() from the Vehicle class.

Did you use the super keyword?
Yes, optionally. If I wanted to include the parent class logic in the overridden method, I used super.methodName() inside the child class method.
Can you call both base and derived methods?
Yes, by using super.methodName() inside the overridden method, or by calling both explicitly in code.
Q3: What happens if you don't override a method?
Main Answer:
If a method is not overridden in the child class, the parent class's version of the method will be used.
Detailed Explanation:
Overriding is optional. If not done, the subclass inherits the method as-is. But if custom behavior is needed (like a truck making a different sound than a car), overriding becomes useful.
Will the base method be called?
Yes, if it's not overridden, the base method will execute by default when called on the subclass object.
Can you make the method optional?
Yes. If the method isn't marked as abstract, the subclass doesn't have to override it.
Can overriding be enforced?
Yes, by marking a method as abstract in an abstract class. Then all subclasses must provide an implementation.
Q4: How is inheritance implemented in TypeScript?
Main Answer:
Inheritance is implemented using the extends keyword, allowing a class to reuse and customize the properties and methods of another class.
Detailed Explanation:
The child class can access all public and protected members of the parent. Private members are not accessible. Constructors can be reused using super().
Example:

```
class Vehicle {
  move() {
    console.log("Vehicle is moving");
  }
}

class Car extends Vehicle {
  move() {
    console.log("Car is driving");
  }
}
```

Here, Car overrides move().
Is TypeScript purely object-oriented?
TypeScript supports object-oriented features like classes, inheritance, interfaces, but it also supports functional programming styles.
How is class syntax compiled to JavaScript?
TypeScript compiles class and inheritance syntax into equivalent JavaScript ES5 or ES6 code based on the target environment.

What access modifiers did you use?
I used public (default), private (hidden from subclasses), and protected (accessible in subclasses but not outside the class).

16. Library System with Modules (TypeScript)

Q1: How did you create modules for Book, User, and Transaction?
Main Answer:
I created separate TypeScript modules by defining each feature (Book, User, Transaction) in its own file and exporting the necessary classes and functions.
Detailed Explanation:
In TypeScript, a file becomes a module when it exports something. For example:

- book.ts contains the Book class and exports it.
- user.ts contains the User class.
- transaction.ts handles borrowing or returning logic.
  Each module was imported wherever needed using import statements. This approach improves code organization and reusability.

What keyword is used to export a class?
The export keyword is used.
Example: export class Book { ... }
How did you manage dependencies?
By importing one module into another using:
import { Book } from './book'
Did you use default exports?
No, I used named exports (export class) so that I can export multiple items from a module if needed.
How are modules organized in folders?
Each module was kept in its own file, often grouped in a modules/ or src/ folder for better structure.
Q2: What is the role of export and import?
Main Answer:
export makes classes, functions, or variables available to other files, while import allows us to use those exported items in another module.
Detailed Explanation:
Without exports, everything in a TypeScript file remains private to that file. By exporting a class like Book, I can then import and use it in a transaction.ts file where book lending logic is written. This helps divide the system into logical parts.
Can you rename imports?
Yes, using the as keyword.
Example:
import { Book as LibraryBook } from './book'
What happens if you import something that doesn't exist?
TypeScript will throw a compile-time error, saying the item is not found in the specified module.
Can you import multiple classes in one statement?
Yes.
Example:
import { Book, User, Transaction } from './librarySystem'

Q3: How did you handle errors like “book not found”?
Main Answer:
I used conditional checks and threw custom errors or displayed messages when a book was not found in the system.
Detailed Explanation:
Before performing an action like issuing or returning a book, I searched the array or collection of books by ID or title. If no matching book was found, I either:

- Threw a custom error (using throw new Error())
- Or returned a message like "Book not available"

Did you use try/catch?
Yes, in areas where exceptions might be thrown (like failed searches or invalid operations).
What is a good way to display errors to the user?
Log them in the console for debugging, and show user-friendly messages in the UI or command line like "Book not available".
How do you throw custom errors?
Using:

```
if (!bookFound) {
  throw new Error("Book not found");
}
```

Q4: Why is modular programming important?
Main Answer:
Modular programming improves code organization, reusability, and maintainability, especially in larger projects.
Detailed Explanation:
By splitting code into modules (like Book, User, Transaction), each part becomes easier to understand, test, and modify. Modules allow multiple developers to work independently on different parts of the app. It also prevents naming conflicts and allows for clear separation of concerns.
How does it help in teamwork?
Each team member can work on a separate module without interfering with others, leading to better collaboration.
What are the benefits of code separation?

- Easier to debug
- Reusable in other projects
- Easier to scale and maintain

Can modules be reused in other projects?
Yes, well-designed modules (like a Book class) can be imported into other applications with similar requirements.

1. AngularJS – *JavaScript Framework for SPAs*

Q: What is AngularJS?
Answer:
AngularJS is a JavaScript framework developed by Google used to build dynamic, single-page web

applications (SPAs). It extends HTML with additional features like directives and enables two-way data binding, which keeps the model (JavaScript data) and the view (HTML) synchronized automatically.

In-depth Explanation:

- Developed by: Google (initial release: 2010)
- Core use: Build responsive web apps where data updates without full page reload
- Two-way data binding: Changes in model instantly reflect in the view, and vice versa
- MVC Pattern: Separates app into Model (data), View (UI), and Controller (logic)
- Key Features:
    - ng-model binds input to data
    - ng-repeat loops over data in templates
    - ng-if, ng-show, ng-hide for conditional rendering
    - Filters like currency, date, uppercase, etc.

Example:

If you're building a product catalog, AngularJS can automatically show the list, update prices, filter by category, and sort them—all without refreshing the page.

Follow-Up Questions with Answers

Q1: What is a single-page application (SPA)?

Answer:

A SPA loads a single HTML page and dynamically updates content using JavaScript. It doesn't reload the whole page on every click, only the required parts change—making it faster and smoother.

Q2: What is two-way data binding?

Answer:

Two-way binding means changes in the UI reflect in the data (model), and changes in the model reflect in the UI—instantly and automatically. For example, typing into an input field updates the data object directly.

Q3: What are directives in AngularJS?

Answer:

Directives are special HTML attributes prefixed with ng-. They add behavior to elements. For example:

- ng-model binds input data
- ng-repeat loops through a list
- ng-click handles clicks

Q4: How is AngularJS different from jQuery?

Answer:

jQuery is mainly used to manipulate the DOM and handle events. AngularJS is a full-fledged framework that handles UI, data binding, routing, and application logic, all in one.

Q5: What is a controller in AngularJS?

Answer:

A controller is a JavaScript function used to manage the data and logic of a particular view. It connects the model (data) and view (HTML template).

Q6: How does routing work in AngularJS?

Answer:

AngularJS uses the ngRoute module. It maps different URLs to different views and controllers. The ng-view directive is used as a placeholder to load the corresponding view dynamically.

Q7: What are filters in AngularJS?

Answer:
Filters are used to format data in the view. Examples:

- currency to show ₹ or $
- uppercase to convert text
- limitTo to limit characters
  They can also be chained for combined effects.

2. React.js – *JavaScript Library for Building User Interfaces*

Q: What is React.js?
Answer:
React.js is a JavaScript library developed by Facebook for building fast, interactive user interfaces, especially in single-page applications (SPAs). It works using a component-based architecture and uses a virtual DOM to update only the parts of the page that change.
In-Depth Explanation:

- Developed by: Facebook (released in 2013)
- Main purpose: Building dynamic and reusable UI components
- Key concepts:
  - JSX: JavaScript + HTML in one file
  - Components: Small, reusable pieces of UI logic (like buttons, cards, forms)
  - Virtual DOM: An in-memory representation of the UI. React updates only what changes, making rendering very fast
  - Unidirectional Data Flow: Data moves from parent to child (props), ensuring predictable behavior
  - Hooks: Special functions like useState, useEffect used to manage component logic

Real-Life Example:
Instagram and Facebook use React. When you like a post or comment, only that part of the UI updates—thanks to React's virtual DOM.
Follow-Up Questions with Answers
Q1: What is JSX in React?
Answer:
JSX stands for JavaScript XML. It allows us to write HTML-like syntax inside JavaScript files. JSX makes the code more readable and looks like regular HTML, but it's compiled to React.createElement() calls behind the scenes.
Example:
<h1>Hello, {user}</h1>

Q2: What is a component in React?
Answer:
A component is a self-contained piece of UI logic. It can be a function or a class, and it returns JSX. React apps are made of many components.
Example:
function Welcome() {

```
  return <h1>Welcome User</h1>;
}
```

Q3: What is the difference between functional and class components?
Answer:

- Class components use ES6 classes and can have lifecycle methods.
- Functional components are simpler, and with React Hooks, they can now handle state and side effects just like class components.

Q4: What is the Virtual DOM?
Answer:
The Virtual DOM is a lightweight copy of the real DOM in memory. When the state of a component changes, React compares the new virtual DOM with the previous one, finds the differences (diffing), and updates only those parts in the real DOM.
Q5: What are props in React?
Answer:
Props (short for "properties") are used to pass data from one component (usually parent) to another (child). Props are read-only.
Q6: What is useState?
Answer:
useState is a React Hook that lets functional components hold and update local state. It returns an array: the current state and a function to update it.
Example:

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

Q7: How is React different from AngularJS?
Answer:

- React is a library, AngularJS is a framework.
- React uses virtual DOM, Angular uses real DOM.
- React has one-way data flow, Angular has two-way binding.
- React focuses only on the view layer, while Angular includes everything (routing, forms, HTTP, etc.)

Q8: What is the state of React?
Answer:
State is a JavaScript object that holds dynamic data in a component. When state changes, the component re-renders to reflect the update.

3. Node.js – *JavaScript on the Server Side*

Q: What is Node.js?
Answer:
Node.js is a runtime environment that allows JavaScript to be run outside the browser, specifically on the server side. It is built on Google Chrome's V8 JavaScript engine and is used to create fast and scalable backend applications, like APIs.

In-Depth Explanation:

- Developed by: Ryan Dahl (2009)
- Built on: Chrome's V8 engine
- Core feature: Non-blocking, event-driven I/O model — ideal for handling many requests at once without waiting
- Uses JavaScript on the server — previously JavaScript was used only on the client side
- Works with files, databases, networking, and APIs
- Handles thousands of simultaneous connections efficiently
- Uses npm (Node Package Manager) to install libraries and frameworks

Real-Life Example:

Node.js is used in real-time apps like chat apps, APIs, stock apps, and online games where performance and speed matter.

Follow-Up Questions with Answers

Q1: What makes Node.js fast and scalable?

Answer:

Node.js uses a non-blocking, event-driven architecture. Instead of waiting for operations like file reading or database access to finish, Node continues executing other tasks and handles the result using callbacks or promises.

Q2: What is npm in Node.js?

Answer:

npm stands for Node Package Manager. It is used to install open-source JavaScript libraries and tools for Node.js.

Example: npm install express installs the Express.js framework.

Q3: What is the difference between Node.js and traditional server languages like PHP or Java?

Answer:

- Node.js is non-blocking, while PHP/Java are mostly blocking (synchronous).
- Node uses one thread for many users, whereas PHP/Java may use a thread per user.
- Node is lightweight and faster for I/O-heavy tasks.

Q4: What is a callback in Node.js?

Answer:

A callback is a function passed as an argument to another function, to be called after a task completes.

Used in asynchronous operations like reading a file or querying a database.

Q5: What are common use-cases of Node.js?

Answer:

- Real-time chat applications
- RESTful APIs
- File uploads
- Streaming services
- IoT apps
- Backend for web and mobile apps

Q6: What is the role of require() in Node.js?

Answer:
require() is used to import built-in or custom modules in Node.js.
Example:
```
const fs = require('fs'); // file system module
```

Q7: Can Node.js interact with databases?
Answer:
Yes, Node.js can interact with both SQL (MySQL, PostgreSQL) and NoSQL databases (like MongoDB) using drivers or ORMs.
Q8: What is Express.js?
Answer:
Express.js is a web framework for Node.js. It simplifies the process of creating RESTful APIs and web applications by providing routing, middleware support, and HTTP handling features.

4. TypeScript – *JavaScript with Types*

Q: What is TypeScript?
Answer:
TypeScript is a superset of JavaScript developed by Microsoft that adds static typing, classes, interfaces, and advanced features to make JavaScript code more robust, readable, and easier to debug—especially in large projects.
In-Depth Explanation:

- Developed by: Microsoft (2012)
- Superset of JavaScript: All JS code is valid TS code
- Adds Features:
    - Static typing (let age: number = 25)
    - Interfaces and type aliases
    - Classes, inheritance, access modifiers
    - Modules, enums, generics
- TypeScript compiles to plain JavaScript, so it runs anywhere JS runs (browser, Node.js)
- Strongly typed: Helps catch errors at compile time, not just runtime
- IDEs like VS Code give better auto-completion and error detection in TypeScript projects

Real-Life Use:
Used in large frontend frameworks like Angular, and scalable projects where type safety and structure are important—like enterprise dashboards, admin panels, or finance apps.
Follow-Up Questions with Answers
Q1: What are the benefits of using TypeScript?
Answer:

- Early error detection (during compilation)
- Better code readability
- IDE support (hints, autocomplete)
- Enforces good coding structure
- Easier refactoring and debugging

Q2: What is the difference between TypeScript and JavaScript?
Answer:

- TypeScript supports static typing, JavaScript does not
- TS has features like interfaces, enums, generics, which JS lacks
- TS needs to be compiled to JS, while JS runs directly

Q3: What is static typing in TypeScript?
Answer:
Static typing means declaring the data type of a variable at the time of declaration. The compiler checks that correct types are used, reducing runtime bugs.
Example:

```
let age: number = 25; // correct
let name: string = "John";
```

Q4: What are interfaces in TypeScript?
Answer:
Interfaces define the structure of an object—which properties and types it should have. It helps enforce consistency across different objects or classes.
Example:

```
interface Person {
  name: string;
  age: number;
}
```

Q5: What are classes in TypeScript?
Answer:
Classes in TS are similar to those in object-oriented languages. You can create objects with properties, constructors, methods, and inheritance.
Example:

```
class Car {
  brand: string;
  constructor(brand: string) {
    this.brand = brand;
  }
  start() {
    console.log(`${this.brand} is starting`);
  }
}
```

Q6: What are access modifiers?
Answer:
TypeScript supports public, private, and protected modifiers to control access to class members:

- public: accessible from anywhere
- private: only inside the class

- protected: in the class and subclasses

Q7: What are generics in TypeScript?
Answer:
Generics allow writing reusable components that work with any type.
Example:
```
function identity<T>(arg: T): T {
  return arg;
}
```

Q8: Can TypeScript be used with React or Node.js?
Answer:
Yes. TypeScript works well with React, Node.js, Express, Angular, and many more. Most major libraries support TypeScript definitions via @types packages.

5. Flask – *Python Web Framework*

Q: What is Flask?
Answer:
Flask is a lightweight, micro web framework written in Python that is used to build web applications and RESTful APIs. It is known for its simplicity, flexibility, and minimal setup, making it ideal for beginners and small-to-medium sized projects.
In-Depth Explanation:

- Developed by: Armin Ronacher (first released in 2010)
- "Micro" means: It does not include built-in tools like authentication or database ORM—but they can be added as needed
- Follows: WSGI (Web Server Gateway Interface) standard and Werkzeug as its core
- Uses: Jinja2 templating engine to generate dynamic HTML pages

Core Concepts:

- Routes are defined using decorators like @app.route('/')
- Runs a web server with Python (app.run())
- Can handle forms, templates, sessions, and APIs
- Used with databases via extensions like Flask-SQLAlchemy

Real-Life Use:
Used in educational projects, dashboards, prototypes, REST APIs, and admin tools.
Follow-Up Questions with Answers
Q1: Why is Flask called a micro-framework?
Answer:
Because it does not come with many built-in features like authentication, ORM, or form validation. Developers can add only the tools they need, which keeps the project lightweight and flexible.
Q2: How do you define a route in Flask?
Answer:
A route maps a URL path to a Python function using a decorator.

Example:

```
@app.route('/')
def home():
    return "Welcome to Flask"
```

Q3: What is the role of app.run() in Flask?

Answer:

app.run() starts the built-in development web server. It allows you to open your Flask app in a browser.

Example:

```
if __name__ == "__main__":
    app.run(debug=True)
```

Q4: What is Jinja2 in Flask?

Answer:

Jinja2 in Flask's template engine that lets you write HTML files with dynamic Python values using curly braces ({{ variable }}) and control structures like loops and conditions.

Example:

```
<h1>Hello, {{ name }}</h1>
```

Q5: How do you create a REST API using Flask?

Answer:

By creating routes that return JSON responses using jsonify(), and using HTTP methods like GET, POST, PUT, and DELETE.

Example:

```
from flask import Flask, jsonify

@app.route('/api/user', methods=['GET'])
def get_user():
    return jsonify({"name": "Alice", "age": 22})
```

Q6: Can Flask connect to a database?

Answer:

Yes. Flask doesn’t come with a database system but supports extensions like:

- Flask-SQLAlchemy for relational DBs
- PyMongo for MongoDB

Q7: What are the advantages of Flask?

Answer:

- Easy to learn and use
- Minimal code to get started
- Flexible structure
- Suitable for quick prototypes
- Good documentation and community

Q8: What are the limitations of Flask?
Answer:

- Not ideal for very large, complex apps out-of-the-box
- Lacks built-in features like user login, admin panels (but they can be added manually)
- Requires more manual setup compared to frameworks like Django

6. AJAX – *Asynchronous JavaScript and XML*

Q: What is AJAX and why is it used?
Answer:
AJAX stands for Asynchronous JavaScript and XML. It allows web applications to send and receive data from a server asynchronously, without reloading the whole web page. This makes websites faster and more dynamic.
In-Depth Explanation:

- Main use: Update parts of a webpage (like search results, suggestions, or form data) without refreshing the page
- Asynchronous: While data is being fetched, the user can still interact with the page
- Works with: JavaScript (or jQuery), HTML, and server-side scripts (PHP, Node.js, Flask, etc.)
- Common data formats: JSON (most preferred), XML, plain text
- Modern AJAX uses: fetch() API or XMLHttpRequest

Real-Life Example:
When you type in Google Search and results appear instantly, that's AJAX at work—it sends your input to the server and updates the suggestions without refreshing the page.
Follow-Up Questions with Answers
Q1: How does AJAX work behind the scenes?
Answer:
AJAX sends an HTTP request in the background using fetch() or XMLHttpRequest. When the server sends back data, JavaScript processes it and updates the HTML using the DOM—without reloading the entire page.

Q2: What is the difference between fetch() and XMLHttpRequest?
Answer:

- fetch() is a modern, promise-based API and easier to use
- XMLHttpRequest is older and uses callbacks
  Both can be used to make GET, POST, PUT, or DELETE requests

Q3: What are the steps to perform an AJAX request?
Answer:

1. Detect user input (e.g., button click or onkeyup)
2. Use fetch() or XMLHttpRequest to send the request
3. Server processes it and sends a response (usually JSON)
4. JavaScript reads the response and updates the page

Q4: What is a real-life use of AJAX in your project?

Answer:
In my product catalog search app, AJAX sends the search query to the backend every time the user types. The results are fetched dynamically and displayed instantly without page reload.
Q5: Is AJAX a programming language?
Answer:
No, AJAX is a technique or approach that combines JavaScript, HTML, and server-side code to make web pages dynamic.
Q6: What are the advantages of using AJAX?
Answer:

- Faster UI updates
- Less bandwidth usage
- Smooth user experience
- Doesn't reload the entire page
- Allows real-time features

Q7: What is the difference between synchronous and asynchronous in AJAX?
Answer:

- Synchronous: Code waits for the server response before continuing (not recommended)
- Asynchronous: Code continues running while waiting for the response—this is default in AJAX and improves performance

Q8: What happens if the server returns an error?
Answer:
In fetch(), the .catch() block is used to handle errors. You can show an error message to the user or retry the request.

7. MongoDB – *NoSQL Database*

Q: What is MongoDB?
Answer:
MongoDB is a NoSQL database that stores data in flexible, JSON-like documents instead of tables like in traditional SQL databases. It is ideal for modern web apps that need to handle large amounts of unstructured or semi-structured data.
In-Depth Explanation:

- Type: NoSQL, document-based database
- Data Format: BSON (Binary JSON) — supports embedded documents and arrays
- Doesn't require a fixed schema — each document can have different fields
- Organized into:
    - Databases → contain
    - Collections → which hold
    - Documents (similar to rows in SQL)
- Commonly used with: Node.js, Express, Python (Flask), Reac
- Works well with REST APIs

Real-Life Example:

Used in web apps like e-commerce platforms where each product can have a different set of properties (size, color, stock), and schema flexibility is needed.

Follow-Up Questions with Answers

Q1: What is the difference between SQL and NoSQL databases?

Answer:

- SQL (like MySQL) uses tables, rows, and fixed schemas
- NoSQL (like MongoDB) uses collections and documents, and doesn't require predefined structure
- NoSQL is more flexible and scalable for large or rapidly changing datasets

Q2: What are the basic CRUD operations in MongoDB?

Answer:

- Create: insertOne(), insertMany()
- Read: find(), findOne()
- Update: updateOne(), updateMany()
- Delete: deleteOne(), deleteMany()

Example:

```
db.users.insertOne({ name: "Alice", age: 22 })
```

Q3: What is a document in MongoDB?

Answer:

A document is a JSON-like object that holds data in key-value pairs.

Example:

```
{ "_id": 1, "name": "John", "email": "john@example.com" }
```

Q4: What is a collection in MongoDB?

Answer:

A collection is a group of related documents, similar to a table in SQL.

Q5: What is BSON?

Answer:

BSON stands for Binary JSON. MongoDB stores data internally in BSON, which is faster to read and write and supports more data types than plain JSON.

Q6: What are indexes in MongoDB?

Answer:

Indexes help speed up search queries. You can create an index on a field (like email) to make lookups faster.

Example: db.users.createIndex({ email: 1 })

Q7: How do you connect MongoDB with Node.js?

Answer:

Using a library like Mongoose or the native mongodb driver.

With Mongoose:

```
mongoose.connect('mongodb://localhost/mydb');
```

Q8: Can MongoDB handle relationships like SQL joins?
Answer:
MongoDB does not have traditional joins, but it supports embedded documents and manual referencing. In some cases, $lookup can simulate a join in aggregation pipelines.

8. RESTful API – *Web Communication via HTTP*

Q: What is a RESTful API?
Answer:
A RESTful API is a web service that follows the REST (Representational State Transfer) principles to allow communication between a client (frontend) and a server (backend) using standard HTTP methods like GET, POST, PUT, and DELETE.
In-Depth Explanation:

- REST was introduced by Roy Fielding in his PhD thesis
- REST is stateless: each request from client to server must contain all the information needed
- REST APIs use URLs (endpoints) to represent resources (e.g., /users, /products)
- Commonly used in frontend-backend apps (like React + Node.js, or Angular + Flask)
- Data is usually transferred in JSON format

HTTP Method to Action Mapping:

Method	Action	Example
GET	Read data	/api/users
POST	Create data	/api/users
PUT	Update data	/api/users/:id
DELETE	Delete data	/api/users/:id

Real-Life Example:
In a travel booking app, the frontend sends a POST request to /api/booking with the user's details to create a new booking. The backend saves it in a database (e.g., MongoDB) and returns a confirmation message.
Follow-Up Questions with Answers
Q1: What does REST stand for?
Answer:
REST stands for Representational State Transfer. It's a set of rules for designing scalable and flexible web services using HTTP.
Q2: What is a resource in the REST API?
Answer:
A resource is any data or object that the API deals with—like user, product, appointment. Each resource is accessed via a unique URL (endpoint).
Example:

- /api/products – all products
- /api/products/123 – product with ID 123

Q3: What does it mean that REST is stateless?
Answer:
Stateless means the server does not store client session data between requests. Each request must contain everything the server needs to understand and respond.
Q4: What data format is usually used in REST APIs?
Answer:
REST APIs most commonly use JSON (JavaScript Object Notation) because it's lightweight, readable, and supported by most languages.
Example response:
{ "name": "Alice", "email": "alice@example.com" }

Q5: How is a REST API different from a SOAP API?
Answer:

- REST is simpler, uses HTTP methods directly
- SOAP is XML-based, heavier, and more complex
- REST is more suitable for modern web and mobile apps

Q6: How do you test REST APIs?
Answer:
Using tools like:

- Postman: to send HTTP requests manually and view responses
- cURL: command-line tool
- Or using frontend JavaScript code (fetch or axios)

Q7: How do you secure REST APIs?
Answer:
Using techniques like:

- Authentication (login system, tokens, JWT)
- HTTPS for encrypted data
- Rate limiting, CORS policies, and input validation

Q8: Can REST API return HTML pages?
Answer:
Yes, but typically REST APIs return data (like JSON), not full HTML pages. The frontend uses this data to dynamically render the content.