100 AngularJS interview questions Answers

Basic Level (1-25)

1. **What is AngularJS?**
    - AngularJS is a JavaScript framework developed by Google to build dynamic web applications. It extends HTML with new attributes and uses a two-way data binding feature.
2. **What are the key features of AngularJS?**
    - Two-way data binding
    - Dependency injection
    - MVC (Model-View-Controller) architecture
    - Directives
    - Filters
    - Templates
    - Routing
3. **What is two-way data binding in AngularJS?**
    - Two-way data binding means that any changes in the UI are immediately reflected in the model and vice versa, eliminating the need for manual DOM manipulation.
4. **What is an AngularJS directive?**
    - Directives are markers on DOM elements that tell AngularJS to attach specific behaviors to elements. Example: `ng-model`, `ng-bind`, `ng-repeat`, etc.
5. **What are the types of directives in AngularJS?**
    - **Attribute directives** (e.g., `ng-model`)
    - **Element directives** (e.g., `<my-directive>`)
    - **Comment directives** (e.g., `<!-- directive: myDirective -->`)
6. **What is the difference between `$scope` and `controller` in AngularJS?**
    - `$scope` is an object that binds the view and controller, while the controller defines the business logic of an application.
7. **What is an AngularJS module?**
    - A module in AngularJS is a container for different parts of an application, such as controllers, services, directives, and filters.

How do you define an AngularJS module?

```
var app = angular.module('myApp', []);
```

8.
9. **What is an AngularJS controller?**

    - A controller is a JavaScript function that controls the application's data and behavior. It is defined inside a module.

How do you define a controller in AngularJS?

```
app.controller('myCtrl', function($scope) {
  $scope.message = "Hello, AngularJS!";
});
```

10.
11. **What is `$scope` in AngularJS?**

    - `$scope` is an object that binds the view (HTML) and the controller. It acts as a bridge between the model and the view.
12. **What is `$rootScope` in AngularJS?**

    - `$rootScope` is a global scope object that is available across all controllers in an AngularJS application.
13. **What is a filter in AngularJS?**

    - A filter is used to format data displayed to the user. Example: `uppercase`, `lowercase`, `currency`, `date`, etc.

How do you apply a filter in AngularJS?

```
{{ name | uppercase }}
```

14.
15. **What is `ng-repeat` in AngularJS?**

    - `ng-repeat` is a directive used to iterate over an array and display data dynamically.

```
<li ng-repeat="item in items">{{ item }}</li>
```

16.
17. **What is `ng-if` in AngularJS?**

    - `ng-if` conditionally renders an element based on a boolean expression.
18. **What is `ng-show` and `ng-hide`?**

    - `ng-show` and `ng-hide` control the visibility of elements based on a condition.

19. **What is `$http` in AngularJS?**

    - `$http` is a service that allows communication with a remote server via HTTP requests.

How do you make an HTTP GET request in AngularJS?

```
$http.get('/api/data').then(function(response) {
  $scope.data = response.data;
});
```

20.
21. **What is `$q` in AngularJS?**

    - `$q` is a service used for handling asynchronous operations (Promises).
22. **What is dependency injection in AngularJS?**

    - Dependency injection (DI) is a design pattern used to inject dependencies (services) into components.
23. **What is `ng-model` in AngularJS?**

    - `ng-model` binds input, select, textarea elements to a property on the `$scope` object.
24. **What is `ng-class` in AngularJS?**

    - `ng-class` dynamically assigns CSS classes to elements based on expressions.
25. **What is `$watch` in AngularJS?**

    - `$watch` monitors changes in variables and executes a function when changes occur.

26.How do you define a custom directive in AngularJS?

```
app.directive('myDirective', function() {
  return {
    template: '<h1>Hello, Directive!</h1>'
  };
});
```

Here are the **AngularJS Intermediate (27-50) and Advanced (51-100) interview questions** along with their answers:

Intermediate Level (27-50)

27. What is `$timeout` and `$interval` in AngularJS?

- `$timeout` is a wrapper around `setTimeout` used to execute a function after a specified delay.
- `$interval` is a wrapper around `setInterval` used to execute a function repeatedly at a fixed time interval.

```
$timeout(function() {
    console.log('Executed after 3 seconds');
}, 3000);

$interval(function() {
    console.log('Executed every 2 seconds');
}, 2000);
```

---

28. What are AngularJS services?

- Services in AngularJS are reusable singleton objects used to share common functionality across controllers, directives, and filters.
- Example: `$http`, `$location`, `$route`, `$timeout`.

---

29. Explain the digest cycle in AngularJS.

- The digest cycle is a process in which AngularJS checks for changes in variables (`$scope`) and updates the DOM.
- It is triggered automatically but can also be manually invoked using `$apply()`.

---

30. What are the different types of scopes in AngularJS?

1. **Global Scope (`$rootScope`)** – Accessible throughout the application.
2. **Controller Scope (`$scope`)** – Available only within the controller.
3. **Isolated Scope** – Used in directives to prevent scope pollution.

---

31. How do you create a factory in AngularJS?

- Factories in AngularJS return an object that contains methods and properties.

```
app.factory('myFactory', function() {
    return {
        greet: function() {
            return "Hello from Factory";
        }
    };
});
```

---

32. What is a singleton service in AngularJS?

- A singleton service is instantiated only once and shared across different components in an application.

---

33. What is `ngRoute` in AngularJS?

- `ngRoute` is a module that allows navigation between views based on URLs.

---

34. How do you implement routing in AngularJS?

- Use `$routeProvider` to configure routes.

```
app.config(function($routeProvider) {
    $routeProvider
        .when('/home', {
            templateUrl: 'home.html',
            controller: 'HomeController'
        })
```

```
    .otherwise({ redirectTo: '/home' });
});
```

---

35. What is `resolve` in AngularJS routing?

- `resolve` ensures that data is loaded before a route is activated.

---

36. How does `$location` service work in AngularJS?

- `$location` allows manipulation of the browser URL within an AngularJS app.

```
$scope.changeURL = function() {
  $location.path('/newRoute');
};
```


---

37. What is lazy loading in AngularJS?

- Lazy loading allows modules, components, and dependencies to be loaded only when required, improving performance.

---

38. What are AngularJS animations?

- Animations are created using the `ngAnimate` module and can be applied to elements using `ng-show`, `ng-hide`, and `ng-class`.

---

39. How do you handle exceptions in AngularJS?

- Using `$exceptionHandler` to catch and log errors.

```
app.factory('$exceptionHandler', function() {
  return function(exception, cause) {
```

```
        console.error(exception.message);
    };
});
```

---

40. What is the purpose of `$on`, `$emit`, and `$broadcast`?

- `$on` – Listens for events.
- `$emit` – Sends an event upwards (parent scopes).
- `$broadcast` – Sends an event downwards (child scopes).

---

41. What are interceptors in AngularJS?

- Interceptors modify HTTP requests/responses globally.

```
app.factory('authInterceptor', function() {
    return {
        request: function(config) {
            config.headers.Authorization = 'Bearer token';
            return config;
        }
    };
});
```


---

42. How does `$compile` work in AngularJS?

- `$compile` is used to manually compile and link HTML templates dynamically.

---

43. What is `$resource` in AngularJS?

- `$resource` simplifies communication with RESTful APIs.

---

44. How do you share data between controllers in AngularJS?

- Using services or `$rootScope`.

---

45. What is `ngSanitize` in AngularJS?

- A module that prevents XSS (cross-site scripting) by sanitizing HTML input.

---

46. How does AngularJS handle security vulnerabilities?

- By using `$sanitize`, escaping input, and avoiding direct DOM manipulation.

---

47. How do you optimize an AngularJS application?

- Minify scripts, enable caching, use `$watch` efficiently, lazy load modules.

---

48. What are AngularJS decorators?

- Functions that modify the behavior of services.

---

49. What is `$parse` in AngularJS?

- `$parse` compiles expressions into functions that can be executed dynamically.

---

50. What is the difference between AngularJS and Angular (2+)?

Feature	AngularJS	Angular (2+)

Architecture	MVC	Component-based
Language	JavaScript	TypeScript
Data Binding	Two-way	One-way (default)
Performance	Slower	Faster
Mobile Support	No	Yes

Advanced Level (51-100)

51. How do you use `$watchGroup` in AngularJS?

- `$watchGroup` watches multiple scope variables at once.

```
$scope.$watchGroup(['var1', 'var2'], function(newValues, oldValues) {
   console.log(newValues);
});
```

---

52. Explain `$evalAsync` in AngularJS.

- `$evalAsync` schedules a task to be executed at the end of the current digest cycle.

---

53. What is `$exceptionHandler` in AngularJS?

- A service to handle errors globally.

---

54. How do you test an AngularJS application?

- Using Karma and Jasmine for unit testing.

---

55. What is `ng-messages` in AngularJS?

- Used for form validation messages.

---

56. How do you handle memory leaks in AngularJS?

- Unsubscribe from `$on` events and remove DOM elements when they are no longer needed.

---

57. What is `$cacheFactory` in AngularJS?

- Used for caching data to improve performance.

---

58. Explain `$animate` in AngularJS.

- `$animate` module adds CSS-based animations to elements.

Here are the **in-depth answers for AngularJS concepts (59-100), including migration, performance tuning, testing, promises, third-party libraries, and best practices.**

---

59. Migration from AngularJS to Angular (2+)

Migrating from AngularJS to Angular (2+) is a multi-step process:

1. **Prepare AngularJS Codebase**
   - Remove unnecessary `$scope` dependencies.
   - Convert AngularJS controllers to components.
2. **Use ngUpgrade**
   - Use `ngUpgrade` to run both AngularJS and Angular code simultaneously.

import { UpgradeModule } from '@angular/upgrade/static';

3.
4. **Migrate Services**
    - Convert AngularJS services to Angular services and use dependency injection.
5. **Convert Directives**
    - Rewrite AngularJS directives as Angular components.
6. **Remove AngularJS Code**
    - Fully transition to Angular and remove AngularJS dependencies.

---

60. Performance Tuning in AngularJS

1. **Use `$watch` efficiently**
    - Minimize the number of watched variables.

```
$scope.$watch('variable', function(newValue) { /* Code */ });
```

2.
3. **Use One-Time Binding**
    - Improve performance by using `::` for static values.

```
<h1>{{ ::title }}</h1>
```

4.
5. **Lazy Load Modules**
    - Load AngularJS modules only when needed.
6. **Optimize DOM Manipulation**
    - Avoid heavy DOM manipulations inside controllers.

---

61. Unit Testing in AngularJS

Unit testing in AngularJS is done using **Karma (test runner)** and **Jasmine (test framework)**.

Install Karma and Jasmine

```
npm install -g karma jasmine
```

1.

Write a simple test

```
describe('TestController', function() {

  beforeEach(module('myApp'));
```

```
    var $controller;

    beforeEach(inject(function(_$controller_) {

        $controller = _$controller_;

    }));


    it('should define a message', function() {

        var $scope = {};

        var controller = $controller('MyCtrl', { $scope: $scope });

        expect($scope.message).toBeDefined();

    });

});
```

2.

Run tests
karma start

3.

---

62. Using `$q` and `defer` for Promises in AngularJS

- `$q` is used for handling asynchronous operations.
- `$q.defer()` creates a deferred object that can be resolved or rejected later.

```
app.service('dataService', function($q, $http) {

    this.getData = function() {

        var deferred = $q.defer();

        $http.get('/api/data')

            .then(function(response) {

                deferred.resolve(response.data);

            })
```

```
        .catch(function(error) {
            deferred.reject(error);
        });
    return deferred.promise;
  };
});
```

---

63. Working with Third-Party Libraries in AngularJS

1. **Include the library**
    - Use CDN or npm to include third-party libraries.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.1/moment.min.js"></script>
```

2.
3. **Use `$window` or Services**
    - Inject the library in a service for better modularity.

```
app.factory('momentService', function($window) {
  return $window.moment;
});
```

4.
5. **Use AngularJS Wrappers**
    - Some libraries provide AngularJS-specific wrappers.

---

64. Custom Directive Best Practices in AngularJS

Use Isolated Scope

```
app.directive('myDirective', function() {
  return {
    scope: { title: '@' },
    template: '<h1>{{ title }}</h1>'
```

```
    };
});
```

1.

Use Controller Instead of Link Function

```
app.directive('myDirective', function() {
    return {
        scope: {},
        controller: function($scope) {
            $scope.message = "Hello from directive!";
        },
        template: '<h1>{{ message }}</h1>'
    };
});
```

2.

Restrict Directives Properly (`E`, `A`, `C`, `M`)

```
restrict: 'E' // Element only
```

3.

65-100. Additional AngularJS In-Depth Concepts

65. Difference between `$broadcast` and `$emit`

- `$broadcast` sends an event downward to child scopes.
- `$emit` sends an event upward to parent scopes.

```
$scope.$broadcast('customEvent', { data: 'test' });
$scope.$emit('customEvent', { data: 'test' });
```

---

66. Best Practices for `$http` in AngularJS

1. Use `.then()` instead of success/error callbacks.
2. Enable caching to reduce redundant API calls.
3. Use interceptors for global request modifications.

```
$http.get('/api/data').then(response => console.log(response.data));
```

---

67. Managing `$digest` Cycle Efficiently

1. Reduce the number of `$watch` functions.
2. Use one-time binding where possible.
3. Manually trigger digest using `$apply()` only when needed.

---

68. Optimizing Directives for Performance

1. Avoid deep-watching objects (`$watch` with `true`).
2. Use `bindToController` instead of `$scope`.
3. Use the `controllerAs` syntax for cleaner code.

---

69. Implementing Caching in AngularJS

- Use `$cacheFactory` to store data in memory.

```
var cache = $cacheFactory('myCache');

cache.put('key', 'value');

console.log(cache.get('key')); // Output: value
```

---

70. AngularJS Memory Management and Preventing Leaks

Remove event listeners when scope is destroyed.

```
 $scope.$on('$destroy', function() {

    element.off();
```

});

1.
2. Avoid excessive `$watch` usage.

---

71. Best Practices for AngularJS Security

1. Enable **CSP (Content Security Policy)**.
2. Sanitize user inputs using `$sanitize`.
3. Prevent **CSRF (Cross-Site Request Forgery)** by adding CSRF tokens in headers.

Here are the **AngularJS interview questions and answers (72-100) covering debugging, performance, server-side rendering, authentication, internationalization, and large-scale application structuring.**

---

72. How to Debug AngularJS Applications Efficiently?

1. **Enable Debug Data**
   - Use `angular.reloadWithDebugInfo()` in the console.

Use `console.log()` and Breakpoints

```
console.log($scope.variable);
```

2.
3. **Use Batarang (Chrome DevTool Extension)**
   - Debug `$scope` variables, performance, and directives.

Enable Strict Dependency Injection

```
angular.module('myApp', []).config(['$provide', function($provide) {
  // Code here
}]);
```

4.

Use `$exceptionHandler` for Error Logging

```
app.factory('$exceptionHandler', function() {
  return function(exception, cause) {
    console.error(exception.message);
  };
});
```

5.

---

73. Using `ngModelOptions` to Optimize Form Input Handling

- **Problem**: By default, `ngModel` updates `$scope` on every keystroke, causing unnecessary digest cycles.
- **Solution**: Use `ngModelOptions` to control update behavior.

```
<input type="text" ng-model="username" ng-model-options="{ updateOn: 'blur' }">
```

- This updates the model only when the input loses focus.

---

74. Implementing Server-Side Rendering in AngularJS Apps

- **Problem**: AngularJS applications render content on the client-side, which can hurt SEO and performance.
- **Solution**: Use tools like **Prerender.io** or **PhantomJS** to generate static content on the server.

Example using **Prerender.io**:

```
 app.config(['$httpProvider', function($httpProvider) {
    $httpProvider.interceptors.push('prerenderInterceptor');
}]);
```

-

---

75. How to Use `$timeout` Properly to Handle Asynchronous Actions?

- `$timeout` ensures that changes are applied after a delay.

Example:

```
 $timeout(function() {
    $scope.message = "Updated after 3 seconds";
}, 3000);
```

-

Best Practice: Always cancel timeouts when scope is destroyed.

```
 var timer = $timeout(function() { /* Action */ }, 5000);
$scope.$on('$destroy', function() {
    $timeout.cancel(timer);
});
```

-

---

76. Structuring Large-Scale AngularJS Applications

1. **Use Modules**
    - Split the app into feature-based modules.

```
angular.module('app.users', []);
angular.module('app.products', []);
```

2.

Use the `controllerAs` Syntax

```
 app.controller('UserController', function() {
    var vm = this;
    vm.name = "John";
});
```

3.
4. **Avoid `$scope` in Controllers**
    - Use services for shared data.
5. **Lazy Load Components**
    - Use `ocLazyLoad` for on-demand loading.

---

77. How to Migrate Legacy AngularJS Projects with Minimal Risk?

1. **Hybrid Approach with `ngUpgrade`**
    - Run AngularJS and Angular together.
2. **Convert Services First**
    - Migrate `$http`-based services to Angular.
3. **Gradually Replace Components**
    - Move from AngularJS controllers to Angular components.
4. **Rewrite Directives as Components**
    - Use Angular's component-based architecture.

---

78. Best Practices for Using Third-Party Authentication Libraries

1. **Use OAuth-based authentication**
    - Firebase, Auth0, Okta, or Passport.js.

Use JWT (JSON Web Token) for Secure Authentication

```
$http.defaults.headers.common.Authorization = 'Bearer ' + token;
```

2.
3. **Store Tokens Securely**
    - Use `sessionStorage` or `HttpOnly Cookies`, **never localStorage**.

---

79. How to Use `$locale` for Internationalization in AngularJS?

Load Locale-Specific Files

```
angular.module('myApp', ['ngLocale']);
```

1.

Use `$locale` Service

```
app.controller('LocaleController', function($scope, $locale) {
  $scope.currency = $locale.NUMBER_FORMATS.CURRENCY_SYM;
});
```

2.

Use `angular-translate` for Language Switching

```
$translate.use('fr');
```

3.

80-100. Other Advanced AngularJS Topics

80. How to Prevent XSS (Cross-Site Scripting) in AngularJS?

Use `$sanitize` module:

```
<div ng-bind-html="trustedHTML | sanitize"></div>
```

•

---

81. How to Optimize Filters for Performance?

- Avoid filtering large datasets inside the template.
- Use **pagination** instead of filtering everything.

---

82. What is the Best Way to Handle Large Lists in AngularJS?

- Use **virtual scrolling** with third-party libraries like `angular-virtual-scroll`.

Use limitTo filter:
<li ng-repeat="item in items | limitTo:50">

-

---

83. How to Implement WebSockets in AngularJS?

Use `$websocket` Service
var ws = $websocket('wss://example.com/socket');
ws.onMessage(function(event) {
  console.log(event.data);
});

1.
2. **Use `socket.io`** for real-time communication.

---

84. How to Improve Routing Performance in AngularJS?

Use `resolve` property to **preload** data before rendering a route.
$routeProvider.when('/dashboard', {
  templateUrl: 'dashboard.html',
  resolve: {
    data: function(DataService) {
      return DataService.getData();
    }
  }
});

-

---

85. How to Secure AngularJS API Calls?

- Always use HTTPS.

- Implement **token-based authentication**.
- Use CSRF protection (`XSRF-TOKEN`).

---

86. What is `$templateCache` in AngularJS?

Stores templates in memory to reduce HTTP requests.
```
$templateCache.put('template.html', '<div>Hello</div>');
```

-

---

87. How to Minimize Digest Cycle Execution Time?

Use `$applyAsync()` instead of `$apply()`.
```
$scope.$applyAsync(function() {
  $scope.data = newData;
});
```

-

---

88. How to Prevent Memory Leaks in AngularJS?

- Use `$destroy` event to clean up resources.
- Unbind event listeners.

---

89. How to Integrate AngularJS with React?

- Use AngularJS directives to wrap React components.

---

90. How to Enable Debugging Mode in AngularJS?

- Use `angular.reloadWithDebugInfo()`.

Here are **questions 91-100** with detailed answers on **advanced AngularJS topics**, including testing, infinite scrolling, scope management, drag-and-drop, and state management. 🚀

---

91. Using `$httpBackend` for Mocking API Calls in Tests

In **unit testing**, `$httpBackend` allows you to **mock HTTP requests** instead of making real API calls.

Example: Mocking an API Response

```
describe('DataService Test', function() {
    var $httpBackend, DataService;

    beforeEach(module('myApp'));
    beforeEach(inject(function(_$httpBackend_, _DataService_) {
        $httpBackend = _$httpBackend_;
        DataService = _DataService_;
    }));

    it('should return mocked data', function() {
        var mockResponse = { name: 'John Doe' };

        $httpBackend.whenGET('/api/user').respond(mockResponse);

        DataService.getUser().then(function(response) {
            expect(response.data).toEqual(mockResponse);
        });

        $httpBackend.flush(); // Triggers the response
    });

    afterEach(function() {
        $httpBackend.verifyNoOutstandingExpectation();
        $httpBackend.verifyNoOutstandingRequest();
    });
});
```


✅ `$httpBackend.flush()` ensures all mocked requests are processed.
✅ Helps in **unit testing** without actual API calls.

---

92. Implementing Infinite Scrolling in AngularJS

Infinite scrolling **loads more data** as the user scrolls down.

Example: Using `ngInfiniteScroll`

Install `ngInfiniteScroll`
npm install ng-infinite-scroll

1.

Add it to your module

```
angular.module('myApp', ['infinite-scroll']);
```

2.

Use it in the template

```
<div ng-repeat="item in items" infinite-scroll="loadMore()">
  {{ item.name }}
</div>
```

3.

Define `loadMore()` in Controller

```
$scope.items = [...]; // Initial data
$scope.loadMore = function() {
  DataService.getMoreItems().then(function(newItems) {
    $scope.items = $scope.items.concat(newItems);
  });
};
```

4.

✅ Ensures a **smooth user experience** without reloading the page.

---

93. Differences Between `$timeout`, `$interval`, and `$digest`

Feature	`$timeout`	`$interval`	`$digest`
Purpose	Executes a function after a delay	Executes repeatedly at a fixed interval	Manually triggers the digest cycle
Use Case	Delayed execution (like `setTimeout`)	Periodic execution (like `setInterval`)	When Angular does not detect changes
Example	`$timeout(fn, 2000);`	`$interval(fn, 1000);`	`$scope.$digest();`

Example Usage

```
$timeout(function() {
  console.log("Executed after 2 seconds");
}, 2000);

var interval = $interval(function() {
  console.log("Repeats every 1 second");
```

```
}, 1000);

$scope.$watch('variable', function(newValue) {
    console.log("Manually triggered digest cycle");
    $scope.$digest();
});
```

✅ `$timeout` and `$interval` **schedule** execution, while `$digest` **forces updates**.

---

94. How to Avoid Scope Pollution in Large Applications?

Scope pollution happens when **too many variables are stored in `$scope`**, causing **memory leaks**.

Best Practices to Prevent Scope Pollution

Use `controllerAs` syntax instead of `$scope`

```
app.controller('MainController', function() {
    var vm = this;
    vm.title = "Hello, World!";
});
<h1>{{ ctrl.title }}</h1>
```

1.

Use Services for Shared Data

```
app.service('UserService', function() {
    this.user = { name: "John Doe" };
});

app.controller('UserCtrl', function(UserService) {
    this.user = UserService.user;
});
```

2.

Destroy `$scope` when Controller Unloads

```
$scope.$on('$destroy', function() {
    $scope.variable = null;
});
```

3.

✅ These practices **improve performance** and **reduce memory leaks**.

---

95. Implementing Drag-and-Drop Features in AngularJS

You can use `ngDragDrop` or HTML5's native **Drag-and-Drop API**.

Example: Using `ngDragDrop`

Install the module

```
npm install angular-dragdrop
```

1.

Include it in the app

```
angular.module('myApp', ['ngDragDrop']);
```

2.

Create Drag and Drop Areas

```
<div ng-drag="true" ng-model="item">Drag Me</div>
<div ng-drop="true" ng-model="droppedItem">Drop Here</div>
```

3.

✅ Provides an easy **drag-and-drop** experience with minimal setup.

---

96. Using Redux-Like State Management in AngularJS

AngularJS doesn't have built-in state management, but you can use **Redux-like patterns**.

Example: Using `angular-redux`

Install Redux for AngularJS

```
npm install @angular-redux/store
```

1.

Create a Store

```
app.factory('store', function() {
  var state = { count: 0 };
  return {
    getState: function() { return state; },
    dispatch: function(action) {
```

```
        if (action.type === 'INCREMENT') {
            state.count++;
        }
    }
  };
});
```

2.

Use Store in a Controller

```
app.controller('CounterCtrl', function($scope, store) {
  $scope.count = store.getState().count;
  $scope.increment = function() {
    store.dispatch({ type: 'INCREMENT' });
    $scope.count = store.getState().count;
  };
});
```

3.

Use in HTML

```
<button ng-click="increment()">Increment</button>
<p>Count: {{ count }}</p>
```

4.

✅ **Manages state globally** like Redux in React.

97-100. Other Advanced Topics

97. How to Handle Large Data Sets Efficiently in AngularJS?

- Use **pagination** instead of loading everything at once.
- Optimize `$watch` and use `track by` in `ng-repeat`.

98. What is the Best Way to Handle User Authentication in AngularJS?

- Use **JWT (JSON Web Tokens)** and store tokens securely in `HttpOnly cookies`.

99. How to Optimize AngularJS for Mobile Performance?

- Avoid heavy DOM manipulations.
- Use **CSS animations** instead of JavaScript.

100. How to Integrate AngularJS with Modern Frameworks?

- Use `ngUpgrade` for migrating to Angular.
- Wrap AngularJS components inside React/Angular Elements.