Here are some **Angular and Frontend Developer interview questions with detailed answers**, tailored for your experience profile (UI/UX and Angular Developer with MEAN exposure):

Angular Interview Q&A

Q1. What are the main features of Angular?
A: Angular offers component-based architecture, two-way data binding, dependency injection, routing, services, directives, and RxJS for reactive programming. These make web applications modular and maintainable.

Q2. What is data binding in Angular?
A: Data binding synchronizes data between the component (TypeScript logic) and the view (HTML). The four main types are:

- Interpolation: `{{ data }}`
- Property binding: `[property]="value"`
- Event binding: `(event)="handler()"`
- Two-way binding: `[(ngModel)]="value"`

Q3. Explain dependency injection in Angular.
A: Dependency injection (DI) allows classes or components to request external dependencies rather than creating them manually. This promotes modularity, easier testing, and reuse. Example: injecting a service into a component using a constructor.

Q4. What are Angular lifecycle hooks?
A: Lifecycle hooks let developers tap into key moments of a component's life. Examples include:

- `ngOnInit()` – runs when initialization is done
- `ngOnChanges()` – detects input property changes
- `ngAfterViewInit()` – executes after view rendering
- `ngOnDestroy()` – used for cleanup and unsubscribing

Q5. How do you handle API integration in Angular?
A: Using Angular's built-in `HttpClientModule`. REST APIs are integrated by making HTTP calls with methods like `GET`, `POST`, `PUT`, `DELETE`, and handling responses or errors using RxJS Observables.

Example:

```typescript
this.http.get('https://api.example.com/data').subscribe(response => {
  this.data = response;
});
```

Frontend Interview Q&A

Q6. How do you ensure website responsiveness?
A: By using CSS Flexbox, Grid layouts, media queries, frameworks like Bootstrap, and testing across devices to ensure designs adapt to various screen sizes.

Q7. What's the difference between '==' and '===' in JavaScript?
A: == checks equality after type conversion, whereas === checks both type and value without conversion, making it the safer option in modern JavaScript.

Q8. How do you optimize frontend performance?
A:

- Minifying CSS/JS files
- Lazy loading modules
- Using OnPush change detection
- Caching static resources
- Compressing images
- Implementing code splitting

Q9. How do you ensure cross-browser compatibility?
A: By testing with tools like BrowserStack, using normalized CSS (like `normalize.css`), avoiding deprecated features, and testing major browsers (Chrome, Edge, Firefox, Safari).

Q10. Explain your experience with MEAN stack projects.
A: I have implemented CRUD operations and developed end-to-end solutions like Library Management and Sales Inquiry Management applications using MEAN. This included Angular for the UI, NodeJS and Express for backend APIs, and MongoDB for efficient data handling.

Here are additional Angular and frontend developer interview questions with answers. This list extends up to 50 questions, covering a wide range of Angular concepts, JavaScript, CSS, and web development best practices:

Angular Interview Questions (1-25)

1. **What is Angular?**
   A: Angular is a platform and framework for building single-page client applications using HTML and TypeScript.
2. **Explain components in Angular.**
   A: Components control views via templates. Each component has a class with properties and methods, an HTML template, and CSS styles.
3. **What is a module in Angular?**
   A: Angular modules organize components, directives, pipes, and services into cohesive blocks of functionality.
4. **What is data binding?**
   A: Synchronizes data between the view and component class. Types are interpolation, property, event, and two-way binding.
5. **What are directives?**
   A: Directives extend HTML with new behavior—structural (ngIf, ngFor) or attribute (ngClass, ngStyle).
6. **Explain services and dependency injection.**
   A: Services provide functionality across components via DI, promoting modular, testable code.
7. **What is RxJS and why is it used?**
   A: RxJS is a reactive programming library for asynchronous events, widely used in Angular for managing data streams.
8. **What is Angular CLI and why use it?**
   A: Command-line tool to generate Angular projects, components, services, and perform builds.
9. **How does Angular handle routing?**
   A: Angular Router provides navigation and URL management for SPA.
10. **What are Angular lifecycle hooks? Provide examples.**
    A: Methods that allow you to tap into component stages; e.g., ngOnInit, ngOnDestroy.
11. **What is change detection?**
    A: Detects and reacts to data changes to update the view efficiently.
12. **Difference between template-driven and reactive forms?**
    A: Template-driven forms are easier but less scalable; reactive forms are more robust and easier to unit test.
13. **Explain Angular pipes.**
    A: Pipes transform data in templates; e.g., date, uppercase, custom pipes.
14. **How to handle HTTP requests?**
    A: Use HttpClientModule; supports GET, POST, PUT, DELETE with observables.
15. **What is lazy loading?**
    A: Loads feature modules on demand improving app load time.
16. **Explain Angular decorators.**
    A: Metadata added to classes (e.g., @Component, @Injectable).
17. **What are guards in Angular?**
    A: Control navigation; include CanActivate, CanDeactivate.
18. **How do you optimize Angular app performance?**
    A: Use OnPush change detection, lazy loading, ahead-of-time compilation.
19. **What is Angular Universal?**
    A: Server-side rendering to improve SEO and initial load time.
20. **Explain the difference between observables and promises.**
    A: Observables can handle multiple events over time; promises handle a single event.
21. **How do you manage state in Angular?**
    A: Using services, @Input/@Output, or state management libraries like NgRx.
22. **How do you secure Angular apps?**
    A: Use authentication (JWT/OAuth), route guards, and sanitize inputs.
23. **What testing frameworks work with Angular?**
    A: Jasmine, Karma, Protractor.
24. **How do you create reusable components?**
    A: Use @Input and @Output to create configurable and event-driven components.
25. **Explain structural directives**
    A: Directives that alter layout by adding/removing elements (e.g., *ngIf, *ngFor)

Frontend Interview Questions (26-50)

26. **What is the difference between HTML and HTML5?**
    A: HTML5 includes semantic tags, audio, video support, and APIs for web apps.
27. **Explain the box model in CSS.**
    A: Every element has content, padding, border, and margin areas.
28. **What is specificity in CSS?**
    A: Rules for which CSS styles apply if multiple rules target the same element.
29. **Difference between class selectors and ID selectors?**
    A: Class selectors can be reused; ID selectors are unique.
30. **What are media queries?**
    A: CSS technique for responsive design based on screen size.
31. **Describe Flexbox.**
    A: Layout module for arranging elements in row/column with flexibility.
32. **How do you optimize website loading time?**
    A: Minify files, lazy load images/scripts, use CDN, leverage browser caching.
33. **Explain event delegation in JavaScript.**
    A: Attaching a single event listener to a parent to handle events on child elements.
34. **What is the DOM?**
    A: Document Object Model, a programming interface for web documents.
35. **Differences between var, let, and const?**
    A: var is function-scoped, let/const are block-scoped; const is immutable.
36. **What is a closure?**
    A: A function retains access to its scope even when executed outside.
37. **Explain promise chaining.**
    A: Using `.then` to sequence asynchronous tasks.
38. **What is event bubbling?**
    A: Events propagate from child to parent elements.
39. **What is the difference between == and ===?**
    A: == compares value after type coercion; === compares value and type.
40. **What is a single-page application?**
    A: Web app that loads a single HTML page and dynamically updates content.
41. **How do you prevent XSS attacks?**
    A: Sanitize inputs, use CSP headers, avoid `innerHTML` assignment.
42. **What are semantic HTML elements?**
    A: Elements that describe their meaning (header, footer, article).
43. **Explain CSS preprocessor.**
    A: Extends CSS with variables, nesting, and functions (e.g., Sass, Less).
44. **What is the shadow DOM?**
    A: Encapsulated DOM subtree to isolate styles/components.
45. **What are web components?**
    A: Reusable custom elements with encapsulated functionality.
46. **Explain how cookies and localStorage work.**
    A: Cookies store small data sent with every HTTP request; localStorage stores larger data client-side.
47. **What is CORS?**
    A: Cross-Origin Resource Sharing, a security feature controlling resource sharing across domains.
48. **How do you handle responsive images?**
    A: Using `<picture>` element or `srcset` attribute.
49. **What is progressive enhancement?**
    A: Build core functionality first, then add advanced features.
50. **Explain the difference between synchronous and asynchronous JavaScript.**
    A: Synchronous executes tasks sequentially; asynchronous handles tasks concurrently.

Here are JavaScript solutions for some common programming problems with both built-in function usage and manual built (custom) implementations:

---

1. Reverse a string

Built-in:

```js
const reverseString = str => str.split('').reverse().join('');
console.log(reverseString("hello")); // "olleh"
```

Built manually:

```js
function reverseStringManual(str) {
  let reversed = '';
  for (let i = str.length - 1; i >= 0; i--) {
    reversed += str[i];
  }
  return reversed;
}
console.log(reverseStringManual("hello")); // "olleh"
```

---

2. Remove duplicates from an array

Built-in:

```js
const removeDuplicates = arr => [...new Set(arr)];
console.log(removeDuplicates([1, 2, 2, 3, 4, 4])); // [1, 2, 3, 4]
```

Built manually:

```js
function removeDuplicatesManual(arr) {
  const unique = [];
  for (let item of arr) {
    if (!unique.includes(item)) {
      unique.push(item);
    }
  }
  return unique;
}
console.log(removeDuplicatesManual([1, 2, 2, 3, 4, 4])); // [1, 2, 3, 4]
```

---

3. Check palindrome

Built-in:

```js
const isPalindrome = str => str === str.split('').reverse().join('');
console.log(isPalindrome("madam")); // true
```

Built manually:

```js
function isPalindromeManual(str) {
  let left = 0, right = str.length - 1;
  while (left < right) {
    if (str[left] !== str[right]) return false;
    left++; right--;
  }
  return true;
}
console.log(isPalindromeManual("madam")); // true
```

---

4. Sum of array numbers

Built-in:

```js
const sumArray = arr => arr.reduce((acc, val) => acc + val, 0);
console.log(sumArray([1, 2, 3, 4])); // 10
```

Built manually:

```js
function sumArrayManual(arr) {
  let sum = 0;
  for (let num of arr) {
    sum += num;
  }
  return sum;
}
console.log(sumArrayManual([1, 2, 3, 4])); // 10
```

---

5. FizzBuzz (print 1 to n)

Built manually (common solution):

```
js
function fizzBuzz(n) {
  for (let i = 1; i <= n; i++) {
    if (i % 15 === 0) console.log("FizzBuzz");
    else if (i % 3 === 0) console.log("Fizz");
    else if (i % 5 === 0) console.log("Buzz");
    else console.log(i);
  }
}
fizzBuzz(15);
```

6. Find the largest number in an array

Built-in:

```
js
const maxNumber = arr => Math.max(...arr);
console.log(maxNumber([1, 5, 3, 9, 2])); // 9
```

Built manually:

```
js
function maxNumberManual(arr) {
  let max = arr[0];
  for (let num of arr) {
    if (num > max) max = num;
  }
  return max;
}
console.log(maxNumberManual([1, 5, 3, 9, 2])); // 9
```

---

7. Find missing number in a sequence (1 to n)

Built manually:

```
js
function findMissingNumber(arr, n) {
  const total = (n * (n + 1)) / 2;
  const sumOfArr = arr.reduce((acc, val) => acc + val, 0);
  return total - sumOfArr;
}
console.log(findMissingNumber([1, 2, 4, 5], 5)); // 3
```

---

8. Flatten nested arrays

Built-in (ES6+):

```Js
const flatArray = arr => arr.flat(Infinity);
console.log(flatArray([1, [2, [3, 4], 5], 6])); // [1, 2, 3, 4, 5, 6]
```

Built manually:

```Js
function flattenManual(arr) {
  let result = [];
  for (let item of arr) {
    if (Array.isArray(item)) {
      result = result.concat(flattenManual(item));
    } else {
      result.push(item);
    }
  }
  return result;
}
console.log(flattenManual([1, [2, [3, 4], 5], 6])); // [1, 2, 3, 4, 5, 6]
```

---

9. Check if two strings are anagrams

Built-in:

```Js
function areAnagrams(str1, str2) {
  return str1.split('').sort().join('') === str2.split('').sort().join('');
}
console.log(areAnagrams("listen", "silent")); // true
```

Built manually:

```Js
function areAnagramsManual(str1, str2) {
  if (str1.length !== str2.length) return false;

  const countChars = (str) => {
    const count = {};
    for (let char of str) {
      count[char] = (count[char] || 0) + 1;
    }
    return count;
  };
  const count1 = countChars(str1);
  const count2 = countChars(str2);

  for (let char in count1) {
    if (count1[char] !== count2[char]) return false;
```

⬇

```
  }
  return true;
}

console.log(areAnagramsManual("listen", "silent")); // true
```

---

10. Validate balanced parentheses

Built manually:

```
js
function isBalanced(str) {
  const stack = [];
  const pairs = { ')' : '(', ']' : '[', '}' : '{' };
  for (let char of str) {
    if ('([{'.includes(char)) stack.push(char);
    else if (')]}'.includes(char)) {
      if (stack.pop() !== pairs[char]) return false;
    }
  }
  return stack.length === 0;
}
console.log(isBalanced("([{}])")); // true
console.log(isBalanced("([{])")); // false
```

Here are more manual star pattern problems in JavaScript with output examples:

---

1. Right-angled triangle star pattern

```
js
function rightAngledTriangle(n) {
  for (let i = 1; i <= n; i++) {
    let line = '';
    for (let j = 1; j <= i; j++) {
      line += '*';
    }
    console.log(line);
  }
}
rightAngledTriangle(5);
```

Output:

```
text
*
**
***
****
*****
```

---

2. Inverted right-angled triangle

```
js
function invertedRightTriangle(n) {
  for (let i = n; i >= 1; i--) {
    let line = '';
    for (let j = 1; j <= i; j++) {
      line += '*';
    }
    console.log(line);
  }
}
invertedRightTriangle(5);
```

Output:

```
text
*****
****
***
**
*
```

---

3. Pyramid star pattern

```
js
function pyramidPattern(n) {
  for (let i = 1; i <= n; i++) {
    let line = '';

    // spaces
    for (let j = 1; j <= n - i; j++) {
      line += ' ';
    }

    // stars with space
    for (let k = 1; k <= i; k++) {
      line += '* ';
    }

    console.log(line);
  }
}
pyramidPattern(5);
```

Output:

```
text
    *
   * *
  * * *
 * * * *
* * * * *
```

---

4. Inverted pyramid star pattern

```
js
function invertedPyramid(n) {
  for (let i = n; i >= 1; i--) {
    let line = '';

    // spaces
    for (let j = 0; j <= n - i - 1; j++) {
      line += ' ';
    }

    // stars with space
    for (let k = 1; k <= i; k++) {
      line += '* ';
    }

    console.log(line);
  }
}
invertedPyramid(5);
```

Output:

```
text
* * * * *
 * * * *
  * * *
   * *
    *
```

---

5. Diamond star pattern

```
js
function diamondPattern(n) {
  // upper pyramid
  for (let i = 1; i <= n; i++) {
    let line = '';
    for (let j = 1; j <= n - i; j++) line += ' ';
    for (let k = 1; k <= i; k++) line += '* ';
    console.log(line);
  }
  // lower inverted pyramid
  for (let i = n - 1; i >= 1; i--) {
    let line = '';
    for (let j = 1; j <= n - i; j++) line += ' ';
    for (let k = 1; k <= i; k++) line += '* ';
    console.log(line);
  }
}
diamondPattern(5);
```

Output:

```
text
    *
   * *
  * * *
 * * * *
* * * * *
 * * * *
  * * *
   * *
    *
```

⬇

Here are some numeric pattern problems solved manually in JavaScript with their outputs:

1. Numeric right-angled triangle pattern

```
js
function numericTriangle(n) {
  for (let i = 1; i <= n; i++) {
    let line = '';
    for (let j = 1; j <= i; j++) {
      line += j + ' ';
    }
    console.log(line);
  }
}
numericTriangle(5);
```

Output:

```
text
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

2. Inverted numeric triangle pattern

```
js
function invertedNumericTriangle(n) {
  for (let i = n; i >= 1; i--) {
    let line = '';
    for (let j = 1; j <= i; j++) {
      line += j + ' ';
    }
    console.log(line);
  }
}
invertedNumericTriangle(5);
```

Output:

```
text
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

3. Floyd’s triangle

```
js
function floydsTriangle(n) {
  let num = 1;
  for (let i = 1; i <= n; i++) {
    let line = '';
    for (let j = 1; j <= i; j++) {
      line += num + ' ';
      num++;
    }
    console.log(line);
  }
}
floydsTriangle(5);
```

Output:

```
text
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

---

4. Square numeric pattern

```
js
function squareNumericPattern(n) {
  for (let i = 1; i <= n; i++) {
    let line = '';
    for (let j = 1; j <= n; j++) {
      line += i + ' ';
    }
    console.log(line);
  }
}
squareNumericPattern(5);
```

Output:

```
text
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

⬇

5. Right-aligned numeric triangle

```
js
function rightAlignedNumericTriangle(n) {
  for (let i = 1; i <= n; i++) {
    let line = '';
    for (let j = 1; j <= n - i; j++) {
      line += '  ';
    }
    for (let k = 1; k <= i; k++) {
      line += k + ' ';
    }
    console.log(line);
  }
}
rightAlignedNumericTriangle(5);
```

Output:

```
text
    1
   1 2
  1 2 3
 1 2 3 4
1 2 3 4 5
```

⬇